****

*** Co je čím ... v počítačových sítích ***

****

Textové soubory jsou ve formátu textového editoru Text602 verze 3.0, obrázky v nekomprimovaném formátu TIF.

dr. ing. Jiří Peterka

katedra softwarového inženýrství

Matematicko-fyzikální fakulta UK

Malostranské náměstí 25

118 00 Praha 1 - Malá Strana

E-mail: PETERKA@KKI.MS.MFF.CUNI.CZ

DCIT, s.r.o.

CDMS, J. Martího 2/407

160 41 Praha 6 - Veleslavín

E-mail: PET@DCIT.CZ

1/ Když se řekne "počítačová síť"

Pod pojmem počítačová síť si snad každý dokáže představit soustavu vzájemně propojených počítačů. Jaký je ale rozdíl mezi "lokální" a "rozlehlou" počítačovou sítí?

Lokální počítačová síť (LAN - Local Area Network),

Rozlehlá počítačová síť (WAN - Wide Area Network).

O přesné formální vymezení těchto dvou pojmů se raději nebudeme pokoušet. Jde totiž o dvě kategorie, do kterých se dnes počítačové sítě rozdělují spíše intuitivním způsobem, neboť mezi nimi neexistují přesně definované hranice. Soustředíme se proto spíše na typické odlišnosti obou kategorií.

Hlavním rozlišujícím kritériem, jak ostatně napovídají slovíčka "lokální" a "rozlehlá", je především geografická oblast, na které jsou jednotlivé uzlové počítače rozmístěny. Lokální počítačová síť se nejčastěji rozkládá v jediné místnosti, v několika místnostech, v jedné budově, případně v rámci několika sousedních budov či např. v rámci areálu univerzity. Vzdálenost mezi jednotlivými uzlovými počítači lokální sítě je tedy nejčastěji v řádu jednotek až stovek metrů, vhodnými technickými prostředky ji ale lze zvýšit i např. na několik kilometrů. Naproti tomu uzlové počítače rozlehlé sítě bývají rozmístěny ve větším regionu, např. v různých městech či dokonce na různých kontinentech.

Další odlišností mezi lokální a rozlehlou počítačovou sítí bývá také druh uzlových počítačů. U lokálních sítí jde obvykle o osobní počítače a tzv. pracovní stanice, tedy o počítače, které jsou určeny vesměs pro jediného uživatele, a jsou v provozu jen tehdy, když je uživatel skutečně potřebuje. Naopak uzlovými počítači rozlehlých sítí bývají nejčastěji počítače z kategorie tzv. střediskových, případně minipočítače, tedy stroje, vybavené sítí terminálů, schopné sloužit většímu počtu uživatelů současně, a pracující nepřetržitě. S tím také souvisí odlišný osud zpráv, které nelze v síti ihned předat jejím adresátům. V rozlehlé síti je zpráva doručena až do příslušného uzlového počítače (který je trvale v provozu) a zde se uchovává až do doby, než si její adresát sedne k

terminálu, přihlásí se do systému a zprávu si převezme. Naopak v lokální síti se zpráva (po upozornění jejího odesilatele) obvykle ztrácí, neboť počítač, na který by měla být doručena, není právě v provozu.

Rozdíl bývá také ve způsobu, jakým jsou uzlové počítače lokální a rozlehlé počítače propojeny. U lokálních sítí si veškerý hardware, potřebný k propojení počítačů (tj. kabeláž, rozbočovače apod.) kupuje majitel resp. provozovatel sítě. Velmi často jsou všechny uzlové počítače připojeny na jediné společné komunikační médium (které pak funguje jako tzv. sběrnice). Efekt je takový, že každý uzlový počítač může mít přímé spojení s kterýmkoli jiným počítačem v lokální síti, a oba počítače tak mohou komunikovat přímo, bez jakýchkoli prostředníků. Naopak u rozlehlých sítí jsou jednotlivé uzlové počítače, rozmístěné ve větších vzdálenostech od sebe, propojeny pomocí přenosových kanálů, které si majitel resp. provozovatel sítě nejčastěji pouze pronajímá od spojových organizací, a nejsou tedy jeho majetkem. Pronájem bývá značně nákladný, proto u rozlehlých sítí není únosné propojovat jednotlivé počítače tak, aby každý z nich mohl mít přímé spojení s kterýmkoli jiným. Dva uzlové počítače rozlehlé sítě tedy mohou být pro vzájemnou komunikaci odkázány na několik prostředníků, tj. na několik mezilehlých uzlových počítačů.

Další odlišností mezi lokální a rozlehlou sítí bývá také účel, ke kterému je síť zřizována. V případě lokálních sítích jde obvykle o možnost sdílení nákladných periferií (např. kvalitních tiskáren, velkokapacitních záložních pamětí) a o možnost sdílení souborů a přístup do společných databází. U rozlehlých sítí bývá hlavním cílem umožnit přenos zpráv a datových souborů na větší vzdálenosti, možnost získat přímý přístup na vzdálený počítač, využití výpočetní kapacity jiného počítače v síti (např. superpočítače), možnost přístupu do rozsáhlých centrálních databází apod.

Z pohledu uživatele však rozlišování mezi lokální a rozlehlou sítí postupně ztrácí význam. Stále častěji totiž dochází k vzájemnému propojování rozlehlých a lokálních sítí. Jedním z mechanismů, kterým se tak děje, je vzájemné propojování jednotlivých lokálních sítí prostřednictvím sítí rozlehlých - lokální sítě zde vlastně začínají vystupovat v roli (koncových) uzlových počítačů rozlehlých sítí. Navzájem se však propojují i jednotlivé rozlehlé sítě, a vzniká celý konglomerát vzájemně propojených sítí, ve kterém se pro uživatele zcela ztrácí rozdíl mezi lokální a rozlehlou sítí - všechny propojené počítače se pro uživatele stávají z hlediska přístupu k síti rovnocenné, a nabízí mu stejné možnosti využití všech zdrojů, služeb a prostředků, které síť resp. celý konglomerát sítí nabízí.

2/ Základní formy přenosů

V počítačových sítích se můžeme setkat s nejrůznějšími formami přenosu signálů, které mohou být navíc různým způsobem modulovány, kódovány, a k jejich přenosu se mohou používat přenosové kanály různých vlastností a charakteristik. Začněme tedy právě u možných forem přenosu signálů:

Paralelní a sériový přenos - Parallel and serial transmission.

Při paralelním přenosu jsou data přenášena po více bitech najednou, typicky po celých bytech. K tomu je ovšem zapotřebí příslušný počet souběžných (paralelních) vodičů, což je únosné jen na krátké vzdálenosti (typicky 20 metrů). S paralelním přenosem se můžeme setkat nejčastěji mezi počítačem a tiskárnou (vybavenou tzv.

paralelním rozhraním, což je standardní případ), v oblasti počítačových sítí pak jen zcela vyjímečně u některých experimentálních lokálních sítí.

Při sériovém přenosu jsou data přenášena postupně bit po bitu, nejnižším (přesněji: nejméně významným) počínaje. V drtivé většině sítí je přenos dat sériový. Nejmenší položka dat přenášená sériově je označována jako **znak (character)** a má obvykle rozsah 7 nebo 8 bitů. Znak, vyjádřený přímo ve formě posloupnosti dvojkových bitů, které se skutečně přenášejí, se pak označuje jako **značka.**

Sériový asynchronní přenos - Serial asynchronous transmission.

Při asynchronním sériovém přenosu mohou být jednotlivé znaky (přesněji: značky) přenášeny s libovolnými časovými odstupy mezi sebou. Příjemce pak ovšem nemůže předem vědět, kdy začíná další znak, a proto musí být schopen jeho příchod podle vhodného příznaku rozpoznat. Tímto příznakem je tzv. **start bit** (též: rozběhový prvek, viz obrázek 2.1.), kterým začíná každý asynchronně přenášený znak. Příchod start bitu je pro příjemce současně i možností správně si nastavit své měřítko času (přesněji: svou časovou základnu). To je nutné proto, aby příjemce správně určil časové okamžiky, kdy má vyhodnocovat stav jednotlivých datových bitů, které po start bitu následují.

Za vlastními datovými bity může následovat jeden tzv. **paritní** bit (viz dále), a konečně tzv. **stop bit** (též: závěrný prvek), jehož délka obvykle odpovídá délce jednoho nebo dvou datových bitů. Stop-bit v sobě nenese žádnou informaci, jeho smyslem je pouze zajistit určitý minimální odstup mezi jednotlivými znaky - vyslání následujícího znaku může začít nejdříve po odvysílání celého předchozího znaku, tedy včetně jeho stop-bitu.

Asynchronnímu způsobu přenosu se někdy říká také **start-stopní** přenos.

.PI OBR02_1.TIF, 20, 40, 10

Sériový synchronní přenos - Serial synchronous transmission.

Při synchronním přenosu jsou obvykle přenášeny celé bloky znaků. Datové bity jednotlivých znaků přitom následují těsně po sobě, bez jakýchkoli časových odstupů, a nejsou prokládány žádnými start či stop bity (mohou však být doplněny jedním paritním bitem). Začátek bloku je indikován jedním nebo několika speciálními synchronizačními znaky (tzv. znaky SYN), jejichž hlavním smyslem je zajistit potřebnou časovou synchronizaci odesilatele i příjemce, tedy pomoci příjemci přesně stanovit časové okamžiky, ve kterých má vyhodnocovat jednotlivé datové bity. Blok znaků je pak opět zakončen synchronizačními znaky, které mohu (ale nemusí) být nepřetržitě vysílány až do začátku následujícího datového bloku.

Synchronní přenos je obecně rychlejší než asynchronní, neboť není zatížen režií připadající na start bity a stop bity. Jeho technická a programová realizace však bývá poněkud složitější než u přenosu asynchronního.

.PI OBR02_2.TIF, 20, 40, 10

Parita - Parity.

Při sériovém i paralelním přenosu dat může docházet k chybám, jejichž důsledkem je přijetí opačné hodnoty jednoho či několika bitů, než jaké byly původně vyslány. Nejjednodušší, ale současně také nejméně účinný způsob zabezpečení znaku, který umožňuje následné rozpoznání výskytu chyby, je doplnění datových bitů jedním dalším bitem tak, aby celkový počet jedniček ve znaku byl (při odesílání) lichý - pak jde o tzv. **lichou paritu (odd parity)** - nebo naopak sudý - pak jde o tzv. **sudá parita (even parity).** Příjemce musí vědět, zda mu odesilatel posílá znaky se sudou či lichou paritou. Pokud počet jedničkových bitů nesouhlasí s očekávanou paritou, může si příjemce dovodit, že došlo k chybě při přenosu jednoho (nebo tří, pěti, obecně lichého počtu) bitů. Má-li přijatý znak očekávanou paritu, není to ještě stoprocentní zárukou jeho bezchybnosti - pomocí jediného paritního bitu nelze rozpoznat chyby v sudém počtu bitů. Zabezpečení pomocí jednoho paritního bitu je tedy vhodné používat jen tam, kde je pravděpodobnost výskytu chyb v jednotlivých bitech malá a pravděpodobnost výskytu chyb ve více bitech současně zanedbatelná.

V praxi se lze setkat také s tím, že se paritní bit nastavuje vždy na 0 (resp. vždy na 1) - v angličtině se tomu říká **space parity** (resp. **mark parity**). Smysl je např. ten, aby odesilatel mohl vysílat sedmibitové znaky doplněné tímto konstantním paritním bitem, které příjemce přijme jako osmibitové znaky bez parity (čímž se ovšem ztrácí možnost detekovat přenosové chyby).

3/ Zabezpečení dat při přenosech

Při přenosech dat může docházet k chybám, a v jejich důsledku může příjemce přijmout jiné znaky, než jaké mu odesilatel původně vyslal. Jedním možným prostředkem pro následnou detekci vzniklých chyb je přidání paritního bitu ke každému přenášenému znaku (jak jsme si naznačili minulý týden) - to je však jen nejjednodušší (a také nejméně účinný) případ použití tzv. **bezpečnostních kódů**.

Základní myšlenka použití bezpečnostních kódů je velmi jednoduchá - původní znaky se podle přesně definovaných pravidel transformují na znaky jiného typu (např. osmibitové znaky se přidáním jednoho paritního bitu převedou na devítibitové). Teprve ty se pak skutečně přenesou a příjemce si je převede zpět do jejich původního tvaru. Některé znaky onoho "jiného typu" však nemohou z původních znaků řádným způsobem nikdy vzniknout (např. při používání liché parity bychom neměli nikdy získat znak se sudou paritou). Pokud pak příjemce přijme takový znak, který při daných pravidlech transformace nemá žádný "vzor", může jej oprávněně považovat za chybně přenesený znak.

Bezpečnostní kódy jsou v zásadě dvojího typu, a to:

detekční kódy - error-detection codes,

které umožňují pouze rozpoznat (detekovat), že přijatý znak je chybný, a

samoopravné kódy - self-correcting codes,

které kromě detekce chyby umožňují i opravu chybně přeneseného znaku, takže jej není nutné přenášet znovu (což u detekčního kódu obecně nutné je).

Použití bezpečnostních kódů vždy znamená, že se v rámci každého znaku ve skutečnosti přenáší více bitů, než kolik by bylo k vyjádření vlastního znaku nezbytně

nutné. Zabezpečení proti chybám není navíc nikdy stoprocentní, jeho účinnost však roste s počtem bitů "navíc". Nejjednodušší detekční kód (zabezpečení sudou nebo lichou paritou) přidává k datovým bitům jeden další bit a dokáže detekovat chybu v jednom bitu. Samoopravný kód, který umožňuje následnou opravu chyby v jediném bitu (tzv. rozšířený Hammingův kód), přidává ke každému 8-bitovému bytu navíc pět bitů (resp. 6 bitů ke každému 16-bitovému slovu).

V praxi je výhodnější nezabezpečovat proti chybám jednotlivé znaky, ale celé posloupnosti znaků resp. celé přenášené bloky dat. Dodatečné bity, používané k detekci chyb, se pak nepřidávají znovu ke každému znaku, ale jen jednou k celému bloku dat (a přenesou se spolu s ním). Je-li pak chyba detekována, nelze ji v rámci bloku lokalizovat až na jednotlivé znaky. Místo toho musí být celý blok prohlášen za chybný a přenesen znovu. To ovšem nemusí být vůbec na závadu - stačí si uvědomit, že přenosy dat téměř vždy probíhají po celých blocích, a nejmenší jednotkou dat, jejíž opakované vyslání si může příjemce vyžádat, je právě celý blok a nikoli jednotlivé znaky.

Podélná parita - longitudinal parity

je jedním možným způsobem zabezpečení celého bloku dat, chápaného jako posloupnost jednotlivých znaků. Zde se nekontroluje sudý resp. lichý počet jedničkových bitů v jednotlivých znacích, ale sudý resp. lichý počet jedničkových bitů ve stejnolehlých bitových pozicích všech znaků v bloku (viz obr. 3.1). Je-li tedy blok dat tvořen např. osmibitovými znaky, přidá se k celému bloku osm paritních bitů (tedy vlastně jeden znak navíc), a každý z nich se nastaví tak, aby byla dodržena sudá resp. lichá parita.

Použití podélné parity se někdy kombinuje i se zabezpečením jednotlivých znaků pomocí sudé resp. liché parity, která se pak pro odlišení od podélné parity označuje jako **příčná** či **znaková parita (transversal, lateral parity).** Obě varianty ilustruje obrázek 3.1.

.PI OBR03_1.TIF, 20, 40, 10

Kontrolní součet - checksum.

Další možností zabezpečení celého bloku dat je součet jednotlivých znaků v bloku, které jsou pro tento účel chápány jako celá dvojková čísla bez znaménka. Kontrolní součet se typicky provádí jako součet modulo 2^8 nebo 2^{16}, tj. výsledkem je kontrolní součet o délce jednoho nebo dvou bytů.

Kontrolní součet i podélnou paritu lze vyhodnocovat průběžně při přijímání jednotlivých znaků bloku. V případě kontrolního součtu se každý nově přijatý znak přičítá ke stávajícímu mezisoučtu, zatímco v případě podélné parity se provádí operace EX-OR (tj. nonekvivalence) jednotlivých bitů nového znaku se stávajícím mezivýsledkem.

Nejúčinnější formu zabezpečení bloku dat však představuje použití tzv.

cyklických kódů - CRC (Cyclic Redundancy Check).

Také zde se podobně jako u výpočtu podélné parity či kontrolního součtu průběžně na základě jednotlivých znaků bloku (přesněji jednotlivých bitů těchto znalů) průběžně vypočítává zabezpečovací údaj. Ten se na konci celého bloku porovná se zabezpečovacím údajem, který podle stejných pravidel vypočítal odesilatel a připojil k odesílanému bloku dat. Pokud se oba údaje shodují, lze přenesený blok s vysokou pravděpodobností považovat za správný - zabezpečení pomocí šestnáctibitového cyklického kódu totiž dokáže spolehlivě odhalit všechny chyby až v šestnácti po sobě jdoucích bitech, a chyby ve větším počtu bitů s přesností 99,9984 %.

Formální důkaz vynikající účinnosti zabezpečení pomocí cyklického kódu sice vyžaduje dosti pokročilý matematický aparát, vlastní způsob výpočtu zabezpečovacího údaje je však až neuvěřitelně jednoduchý (bohužel však zde již nemáme potřebný prostor k tomu, abychom tuto jednoduchost mohli patřičně "vychutnat"). Stačí k němu jednoduchý posuvný registr, umožňující provést operaci EX-OR (tj. nonekvivalenci jednotlivých bitů) s pevně danou maskou. Hodnota této masky je jednoznačně určena tzv. **generujícím polynomem (generating polynomial)**, na kterém musí být příjemce i odesilatel předem dohodnuti. Použitelných tvarů těchto polynomů je více; v oblasti komunikací se nejčastěji používá polynom $x^{16} + x^{12} + x^5 + 1$, doporučený organizací CCITT.

4/ Modulace

Potřebujeme-li přenášet dvojková data po signálových vodičích, můžeme obě možné hodnoty, 0 a 1, reprezentovat pomocí úrovní napětí na vodiči - např. podle obr. 4.1 a/ jednou nulovou a jednou nenulovou úrovní, či podle obr. 4.1. b/ jednou zápornou a jednou nezápornou úrovní. Používají se ovšem i složitější způsoby vyjádření logických hodnot pomocí úrovní napětí - příkladem může být tzv. kód Manchester II (viz obr. 4.1. c/), který se používá u lokálních sítí Ethernet, a zajišťuje určitý minimální počet změn přenášeného signálu i v případě, že má být přenášena delší posloupnost bitů stejné hodnoty (např. dlouhá řada nul). Všechny tyto způsoby přenosu jsou souhrnně označovány jako

přenosy v základním pásmu - baseband transmissions.

Problém je však v tom, že mnohé přenosové cesty (např. běžné telefonní okruhy apod.) jsou vzhledem ke svým fyzikálním vlastnostem pro přenos v základním pásmu prakticky nepoužitelné, zatímco jiná média (např. koaxiální kabely) sice pro přenos v základním pásmu můžeme využít, ale nikoli s maximální možnou efektivitou.

.PI OBR04_1.TIF, 20, 40, 10

Alternativou k přenosu v základním pásmu je

přenos v přeloženém pásmu - broadband transmission

při kterém je přenášen takový signál, který se daným přenosovým médiem šíří nejlépe (s nejmenšími ztrátami). Typicky jde o pravidelně se měnící signál sinusového průběhu (tzv. harmonický signál), který ukazuje obrázek 4.2 a/. Užitečná informace se

pak přenáší prostřednictvím změn v průběhu tohoto signálu. Lze si představit, že harmonický signál je jakýmsi nosičem (proto se mu takéříká **nosný signál** resp. **nosná,** anglicky **carrier)**, a užitečná informace se na něj "nanáší" postupem označovaným jako

modulace - modulation.

.PI OBR04_2.TIF, 20, 40, 10

Existují různé možnosti modulace nosného signálu:

amplitudová modulace - amplitude modulation (AM),

při které jsou jednotlivé logické hodnoty vyjádřeny různými hodnotami amplitudy (rozkmitu) harmonického signálu - viz obr. 4.2. b/,

frekvenční modulace - frequency modulation (FM),

při které jsou jednotlivé logické hodnoty vyjádřeny různými frekvencemi (kmitočty) harmonického signálu - viz obr. 4.2 c/,

fázová modulace - phase modulation (PM),

při které jsou jednotlivé logické hodnoty vyjádřeny různou fází (posunutím) harmonického signálu - viz obr. 4.2 d/.

Nosný signál, používaný při přenosech v přeloženém pásmu, je vždy

analogovým signálem - analog signal,

tedy signálem, který může nabývat spojité množiny různých hodnot, tj. měnit se spojitě. Příkladem může být právě harmonický signál dle obr. 4.2. Naproti tomu

číslicový, diskrétní signál - digital signal

může nabývat jen konečně mnoha různých hodnot (např. jen dvou, jako na obrázku 4.1.) a mění se skokem.

Modulací vzniká z analogového nosného signálu opět analogový signál. Musí však být možné u něho rozlišit potřebný počet navzájem různých stavů, které mohou reprezentovat diskrétní logické hodnoty. Pokud modulací vznikají jen dva navzájem rozlišitelné stavy nosného signálu (jako např. při fázové modulaci posunutím signálu o 0 stupňů a o 180 stupňů), jde o modulaci tzv. dvoustavovou, která nese pouze jednobitovou informaci, neboť dva rozlišitelné stavy nosného signálu mohou reprezentovat zase jen dvě diskrétní logické hodnoty. Používá se však i modulace s větším počtem navzájem rozlišitelných stavů nosného signálu. Např. při čtyřstavové fázové modulaci s posunutím fáze nosného signálu o 0, 90, 180 a 270 stupňů může jeden stav nosného signálu reprezentovat jednu ze čtyř možných logických hodnot, a tedy nést dvoubitovou informaci. V praxi se jednotlivé způsoby modulace navzájem kombinují - např. v telefonních modemech pro vyšší přenosové rychlosti se kombinuje fázová modulace s modulací amplitudovou. Cílem je totiž zvětšit počet rozlišitelných stavů nosného signálu, který tak může nést vícebitovou informaci.

Nyní si již můžeme snadno vysvětlit rozdíl mezi modulační a přenosovou rychlostí:

Modulační rychlost - modulation speed

vyjadřuje počet změn nosného signálu za jednotku času (sekundu), a měří ze v Baudech (zkratkou Bd). Modulační ryhlost ještě neříká nic o tom, jaké množství informace nosný signál přenáší.

Přenosová rychlost - transmission speed

naopak udává objem informace, přenesený za časovou jednotku. Vyjadřuje se v bitech za sekundu (bits per second, zkratkou bps). Přenosová rychlost naopak neříká nic o tom, jak rychle se mění nosný signál.

Modulační rychlost může být rovna rychlosti přenosové, a to právě v případě dvoustavové modulace. Pokud ale používáme např. modulaci čtyřstavovou, vyjadřuje jeden stav nosného signálu dvoubitovou informaci a přenosová rychlost je pak číselně dvojnásobná oproti rychlosti modulační.

5/ Šířka pásma a její dělení

Při přenosu informací je jedním z rozhodujících aspektů objem dat, který je používaný přenosový kanál schopen přenést za určitý čas. Obvykle se v této souvislosti mluví (spíše neformálně) o přenosové kapacitě či propustnosti přenosové cesty. Správným měřítkem je však pouze přenosová rychlost (v bitech za sekundu), kterou jsme se zabývali minulý týden.

Dosažitelná přenosová rychlost je ale vždy dána souhrnem fyzikálních vlastností přenosového média (vodičů, kabelů apod.) a vlastnostmi dalších technických prostředků, které přenosový kanál spoluvytvářejí (např. modemů, multiplexorů apod.).

Každý přenosový kanál je vždy schopen přenášet jen signály o frekvenci z určitého omezeného intervalu. Přesněji: signály s jinou frekvencí přenáší tak špatně (s tak velkým útlumem, zkreslením apod.), že není únosné jej pro přenos těchto signálů vůbec používat. Například běžné telefonní okruhy jsou schopné přenášet signály s frekvencí přibližně od 300 do 3400 Hz.

Šířka intervalu frekvencí, které je přenosový kanál schopen přenést, představuje tzv.

šířku pásma - bandwidth.

Jednotka šířky pásma je stejná jako jednotka frekvence, tj. 1 Hz. V případě běžných telefonních okruhů, schopných přenášet frekvence od 300 Hz do 3400 Hz, je tedy šířka pásma 3100 Hz, tj. 3,1 kHz.

Obecně platí, že čím větší je šířka pásma přenosového kanálu, tím větší je přenosová rychlost, kterou na něm lze dosáhnout.

Přesnou závislost mezi dosažitelnou přenosovou rychlostí a dostupnou šířkou pásma však nelze jednoduše stanovit - velmi totiž záleží na konkrétní realizaci. Existují však teoretické výsledky, které poskytují horní odhad této závislosti. Konkrétně stanovují maximální teoreticky dosažitelnou modulační i přenosovou rychlost při dané šířce pásma přenosového kanálu. V případě modulační rychlosti (tedy počtu změn nosného signálu za jednotku času, viz minulý týden) je vzájemná závislost velmi jednoduchá - maximální modulační rychlost je číselně dvojnásobkem šířky pásma. Také maximální dosažitelná přenosová rychlost je číselně přímo úměrná šířce pásma - konstanta úměrnosti je však závislá na "kvalitě" přenášeného signálu (přesněji na odstupu užitečného signálu od šumu). Například pro odstup signál/šum 30 dB (což znamená, že užitečný signál je 1000-krát silnější než šum) má konstanta úměrnosti hodnotu

přibližně 9,96. Při šířce pásma telefonního okruhu 3,1 kHz by to znamenalo maximální přenosovou rychlost přes 30000 bitů za sekundu. Tuto hodnotu je ovšem nutné chápat skutečně jen jako teoretický horní limit, který se v praxi ani zdaleka nedosahuje. Například právě na běžných telefonních okruzích se dnes s nejkvalitnějšími modemy dosahují přenosové rychlosti kolem 14400 bitů za sekundu.

Vedle telefonních okruhů samozřejmě existují i jiné druhy přenosových kanálů, jejichž šířka pásma je výrazně vyšší, a vyšší je pak také přenosová rychlost, která je na nich reálně dosažitelná (v dalších pokračováních se o nich zmíníme podrobněji). Zde pak může být otázkou, jak celkovou přenosovou kapacitu skutečně využít, potřebujeme-li například jen určitou (řádově menší) přenosovou rychlost, zato ale pro větší počet na sobě nezávislých uživatelů.

Existuje technika, které se v angličtině říká

multiplexing

a která umožňuje rozdělit jeden přenosový kanál s velkou šířku pásma na několik (užších) logických subkanálů, které se ovšem jeví jako samostatné, na sobě nezávislé přenosové kanály. Technické zařízení, které takovéto logické rozdělení na několik subkanálů zajišťuje, se nazývá

multiplexor - multiplexer.

Existují dva základní způsoby dělení jednoho přenosového kanálu na více subkanálů**.** Prvním z nich je tzv.

frekvenční multiplex - frequency division multiplexing (FDM).

.PI OBR05_1.TIF, 20, 40, 10

Zde si lze představit, že jednotlivé subkanály jsou "navršeny na sebe" v přenosovém pásmu skutečně existujícího přenosového kanálu, a každému z nich je přidělena taková část celkové šířky pásma, jakou potřebuje (tj. jaká je jeho šířka pásma) - viz obr. 5.1. Signál, přenášený v rámci určitého subkanálu, musí multiplexor nejprve frekvenčně "posunout" do části pásma, přidělené danému subkanálu, a na druhé straně spoje jej zase "vrátit zpět" do původní frekvenční polohy. Celý mechanismus je přitom plně transparentní, tj. uživatelé jednotlivých kanálů si mohou myslet, že mají k dispozici samostatné, na sobě nezávislé přenosové kanály.

Druhou základní možností pro dělení jednoho přenosového kanálu na více subkanálů je tzv.

časový multiplex - time division multiplexing (TDM).

Zde je vlastní přenosový kanál pravidelně přidělován s celou svou šířkou pásma na krátké časové intervaly jednotlivým subkanálům. Nejsnáze se tato představa ilustruje na příkladu kanálu, který přenáší přímo číslicová data. Multiplexor nejprve "vybere" například po jednom bitu od každého subkanálu, a ze všech těchto bitů sestaví vícebitový znak, který přenese kanálem. Na opačné straně kanálu pak druhý multiplexor (někdy označovaný jako demultiplexor) rozebere přijatý znak na jednotlivé bity a ty předá příslušným subkanálům - viz obr. 5.2.

.PI OBR05_2.TIF, 20, 40, 10 .cp20

Při časovém i frekvenčním multiplexu samozřejmě musí platit, že součet šířek pásma jednotlivých subkanálů musí být menší než celková šířka pásma existujícího přenosového kanálu. Časový multiplex je obecně účinnější, v tom smyslu, že součet šířek pásma subkanálů může být "blíže" teoretické horní hranici, tedy celkové šířce pásma existujícího kanálu.

6/ Kanál, okruh, spoj, přenosová cesta

Minulý týden jsme začali operovat s pojmy "kanál" a "okruh", aniž jsme si je přesněji vymezili. Jde o pojmy, které intuitivně často splývají, a v praxi se mnohokráte zaměňují i s dalšími termíny, jako např. "přenosová cesta" či "spoj", aniž by tím srozumitelnost populárně laděných textů výrazněji utrpěla.

Jistě ale neuškodí, když se pokusíme o rozlišení obsahu těchto pojmů i dalších souvisejících termínů - tím si současně naznačíme další zajímavé souvislosti.

Potřebujeme-li odněkud někam přenést nějaká data, můžeme to udělat také tak, že je nahrajeme např. na diskety či na magnetické pásky, a ty pak fyzicky přeneseme z jednoho místa na druhé. Pak je asi správnější hovořit spíše o **přepravě dat** než o jejich přenosu. Mezi zdrojem dat a místem jejich určení tak vzniká vazba, které si zaslouží přívlastek

nespřažená - off-line,

vystihující právě tu skutečnost, že vzájemná komunikace se uskutečňuje fyzickým přenosem vhodných nosičů dat.

Existuje-li však mezi zdrojem a příjemcem možnost přenosu elektronickou cestou, pak je na místě mluvit o přenosu dat, a používat pro něj přívlastek

spřažený - on-line.

Nutnou podmínkou přenosu elektronickou cestou je vhodný

kanál - channel,

čímž se chápe souhrn prostředků, které vytváří telekomunikační spojení dvou míst. Každý kanál je charakterizován určitou šířkou pásma, přenosovou rychlostí, úrovní šumu, chybovostí atd.

Kanál je ale obvykle chápán jako jednosměrný. V případě možnosti obousměrného spojení se používá spíše termín

okruh - circuit,

pod kterým si můžeme představit dvojici jednosměrných kanálů.

Pro vytvoření kanálu či okruhu musí existovat vhodná

přenosová cesta - transmission path,

která zajišťuje vlastní přenos signálů - může jít např. o různé druhy kabelů či světlovodů (optických vláken) - pak jde o **linkové** (resp. drátové) přenosové cesty, nebo také prostředky krátkovlnné, radioreléové, družicové apod. - pak jde o přenosové cesty **radiové** (resp. bezdrátové).

Schopnost každé přenosové cesty přenášet signály resp. data charakterizuje její šířka pásma. Ta může být využita pro jediný kanál (o stejné šířce pásma), nebo může být

rozdělena mezi několik kanálů či okruhů (o menších šířkách pásma) například pomocí časového či frekvenčního multiplexu, jak jsme si ukazovali minulý týden. Příkladem mohou být družicové spoje, které umožňují zřídit v rámci jediné přenosové cesty řádově tisíce telefonních okruhů.

Je ovšem možný i opačný případ - pokud je třeba vytvořit kanál či okruh o větší šířce pásma, než jakou mají jednotlivé dostupné přenosové cesty, lze pomocí vhodných technických prostředků vytvořit jeden kanál či okruh pomocí více přenosových cest.

Kanál a okruh představují spojení dvou míst. Pokud mezi těmito dvěma místy vede vhodná přenosová cesta, je možné vytvořit kanál či okruh pomocí této jediné přenosové cesty. V opačném případě je nutné vytvořit kanál či okruh pomocí více přenosových cest, které na sebe navazují. Například kanál či okruh, spojující dvě místa na různých kontinentech, může být veden po souši např. podzemními kabely (tj. linkovými přenosovými cestami), a přes oceán např. přes družici (tedy bezdrátovou přenosovou cestou). Uživatel však tuto skutečnost nepozná, neboť využívá kanál resp. okruh, a ten se mu jeví jako přímé spojení obou míst.

Přenosová cesta je tedy pojmem, který se týká spíše struktury trvale existujících spojových prostředků, zatímco pojmy kanál a okruh v sobě zahrnují i jejich organizaci, tedy způsob jejich využití. Jak uvidíme příště, kanály i okruhy mohou být zřizovány i dočasně, teprve tehdy, až skutečně vznikne potřeba spojení dvou míst, a poté zase zanikat. Pro přenosové cesty tato možnost již neplatí.

Samotné kanály a okruhy však ještě nejsou schopny poskytovat potřebné přenosové služby. Např. telefonní okruh je sice schopen přenášet telefonní signály, ale k jeho praktickému využití potřebujeme nutně i telefonní přístroje, které převádí lidský hlas na telefonní signál a naopak. Teprve telefonní okruh, vybavený na obou koncích telefonními přístroji, vytváří tzv.

spoj - link,

v tomto případě telefonní spoj. Podobně např. telegrafní okruh, zakončený na obou koncích telegrafním či dálnopisným přístrojem, vytváří telegrafní spoj.

Existují však i přenosové cesty, které jsou současně i spoji. Jde např. o radioreléový spoj, který již je vybaven koncovým zařízením umožňujícím bezdrátový přenos, nebo družicový spoj, vybavený koncovými pozemskými stanicemi a mezilehlou družicovou stanicí.

V oblasti počítačových sítí jsou samozřejmě nejzajímavější takové spoje, které slouží k přenosu dat. Pak jde o tzv.

datové spoje - data links,

kterými se budeme ještě zabývat podrobněji.

Na závěr si však uvedeme ještě jeden neformální pojem, který se běžně používá v rozhovorech mezi lidmi a v populárních textech. Jde o pojem

linka,

pod kterým se obvykle skrývá to, co by mělo být přesněji označeno jako okruh, případně jako kanál. V angličtině se podobným způsobem používá termín

line,

který v užším slova smyslu znamená spíše linkovou přenosovou cestu.

7/ Klasifikace okruhů

Počítačové sítě, zvláště pak sítě rozlehlé, používají ke své činnosti různé druhy okruhů.

Jedním možným kritériem pro klasifikaci používaných okruhů může být vztah mezi jejich vlastníkem a jejich uživatelem. Pokud si uživatel zřídí potřebnou přenosovou cestu sám (např. natáhne či položí potřebné vedení mezi dvěma blízkými uzlovými počítači) a na ní si zřídí okruh, jde o

soukromý okruh - private circuit.

V každém státě vždy existuje spojová organizace, která zřizuje

veřejné okruhy - public circuits,

které může za patřičný poplatek (a při dodržení příslušných technických podmínek) používat každý - příkladem může být veřejná telefonní síť.

Spojové organizace (správy spojů) mají obvykle nejlepší podmínky pro budování a provozování nejrůznějších přenosových cest a spojových prostředků (a obvykle i legislativně vytvořený monopol na tuto činnost). Provozovatelé počítačových sítí, kteří potřebují mít vhodný okruh trvale k dispozici a jen pro sebe, si jej pak musí nechat od spojové organizace za patřičný poplatek zřídit (vytvořit) a pronajmout. Jde pak o

pronajatý okruh - leased circuit,

který zůstává majetkem spojové organizace (správy spojů), ale výhradní právo na jeho využití má ten, kdo si jej pronajal. Naše správa spojů nazývá tento druh okruhů příznačně "propůjčenými" okruhy.

Dalším možným kritériem pro dělení spojů je jejich trvalá či dočasná existence. Vezměme si jako příklad veřejnou telefonní síť. Zde si jistě každý dokáže představit, že z každého telefonního přístroje vede dvojice vodičů (tedy linková přenosová cesta) jen do nejbližší telefonní ústředny. Chceme-li se spojit s jiným telefonním účastníkem, který je obdobným způsobem připojen k téže telefonní ústředně, musí uvnitř této ústředny dojít k dočasnému propojení obou vedení tak, aby mezi oběma účastníky vzniknul potřebný telefonní okruh. Po ukončení hovoru pak tento dočasný okruh zaniká a v rámci ústředny se obě přenosové cesty mohou rozpojit. Obdobně je tomu i v případě, že oba účastníci telefonního styku jsou připojeni k různým ústřednám - zde se pro vytvoření dočasného telefonního okruhu musí vhodným způsobem využít ještě i přenosové cesty vedoucí mezi oběma ústřednami - vždy ovšem dočasně, jen po dobu konání hovoru.

Takovéto spojení mezi dvěma místy, které neexistuje trvale v čase, ale je pouze dočasně sestavováno až v případě skutečné potřeby, představuje tzv.

komutovaný okruh - switched circuit, dial-up circuit.

Komutovaný proto, že procesu přepojování uvnitř ústředny se odborně říká **komutace.**

Naopak spoj, který existuje trvale (i v době, kdy není případně využíván), se nazývá

pevný okruh - non-switched circuit, direct circuit.

Okruhy, které si uživatelé pronajímají od spojových organizací (tj. pronajaté okruhy), jsou vždy pevnými okruhy. Zřízení pevného okruhu obvykle znamená, že v rámci telefonní ústředny se napevno propojí vhodné, již existující přenosové cesty (resp. na nich vytvořené okruhy). Pevný okruh přitom může procházet přes více telefonních ústředen, v každé z nich jsou však příslušné přenosové cesty resp. okruhy vždy spojeny napevno. Pevný okruh tak obchází přepojovací zařízení v telefonních ústřednách, která jsou obvykle zdrojem nejrůznějších poruch a rušení na komutovaných okruzích. To je jedním z důvodů, proč mohou být pevné okruhy kvalitnější, než okruhy komutované (a v důsledku toho mohou umožňovat dosažení vyšších přensoových rychlostí). Dalším důvodem pro vyšší kvalitu pevných okruhů je možnost vybrat pro ně dostatečně kvalitní přenosové cesty již při jejich zřizování.

Spojové organizace zřizují pevné okruhy vždy až na základě požadavků uživatelů. Podle svých technických možností pro takové přenosové rychlosti, jaké si uživatel objedná (a samozřejmě také zaplatí). Pro rozlehlé počítačové sítě se dnes ve světě používají pronajaté pevné okruhy nejčastěji s přenosovou rychlostí 64 kbit/sekundu a vyšší. V našich podmínkách jsou však zatím běžnější spíše spoje s nižšími přenosovými rychlostmi.

Další možné hledisko pro klasifikaci okruhů je to, zda jsou určeny pro přenos analogových signálů (např. telefonních, rozhlasových či televizních signálů) , tj. jde-li o

analogové okruhy - analog circuits,

.cp9

nebo jsou-li určeny pro přenos číslicových signálů, tj. jde-li o

číslicové okruhy - digital circuits.

Je dobré si uvědomit, že jednotlivá kritéria klasifikace se navzájem nemusí vylučovat. Veřejné komutované analogové okruhy si asi dokáže představit každý - příkladem může být právě veřejná telefonní síť. Pronajaté okruhy bývají pevné (nikoli komutované), ale mohou být jak analogové, tak i číslicové. Ve světě dnes existují i číslicové okruhy, které jsou veřejné a komutované.

8/ Datové okruhy a spoje

U lokálních počítačových sítí si potřebné prostředky pro vzájemné propojení uzlových počítačů zajišťuje a provozuje nejčastěji uživatel sám. Vzhledem k lokalizaci celé sítě (např. v rámci jediné budovy, která je vlastnictvím uživatele) tím totiž neporušuje obvyklý monopol spojových organizací (správ spojů) na vlastnictví přenosových cesty a poskytování přenosových služeb, který se obvykle týká jen větších vzdáleností a veřejných objektů a prostranství. Provozovatelé resp. zřizovatelé lokálních sítí si tedy mohou sami zvolit a zrealizovat takový způsob propojení svých uzlových počítačů, jaký jim nejlépe vyhovuje - nejčastěji pomocí kroucené dvoulinky, koaxiálního kabelu či dokonce pomocí optických vláken, tedy vesměs pomocí linkových přenosových cest, schopných přímo přenášet číslicové signály.

V případě rozlehlých počítačových sítí jsou jejich zřizovatelé v poněkud jiné pozici, neboť technické prostředky pro propojení svých uzlových počítačů si musí pronajmout od příslušných spojových organizací. Je celkem pochopitelné, že tyto spojové organizace přitom budou určovat pravidla hry - tedy přesné technické i další podmínky, které musí uživatel dodržovat, chce-li pronajatý majetek správy spojů používat.

Spojová organizace (správa spojů) uživateli poskytuje telekomunikační kanál či okruh s určitými technickými parametry, které definují, jaké signály je schopen přenášet. Uživatel ovšem může mít jiné požadavky na přenosové schopnosti okruhu, po kterém potřebuje přenášet signály jiných parametrů. Proto se na oba konce okruhu připojuje zařízení, které potřebné úpravy signálu zajišťuje (viz obr. 8.1.). Toto zařízení, fungující jako měnič signálu, se v odborné terminologii nazývá

ukončujícím zařízením datového okruhu (UZD) - data circuit terminating equipment (DCE).

Uvažujme následující typický příklad: přenosový okruh nechť je běžným komutovaným telefonním okruhem, tedy okruhem analogovým, schopným přenášet analogové signály v rozmezí 300 až 3400 Hz. My ho ovšem potřebujeme využívat pro přenos číslicových dat, tedy jako okruh číslicový. Již dříve jsme si ukázali, že takovýto postup je možný - pomocí modulace. Místo diskrétního číslicového signálu tedy budeme přenášet vhodný analogový signál, který je telefonní okruh schopen přenést, a který budeme modulovat (tj. měnit jeho průběh) podle toho, jaká data (resp. číslicový signál, který je vyjadřuje) chceme přenést. Potřebujeme k tomu ale takové zařízení, které je schopno zajistit jednak potřebnou modulaci analogového signálu, jednak i opačný proces, tzv. **demodulaci**, tedy zpětné získání číslicového signálu ze signálu analogového. Toto zařízení se nazývá

modem,

což je zkratka od **MO**dulátor/**DEM**odulátor.

Ukončujícím zařízením datového okruhu (zařízením UZD) bývá nejčastěji právě modem, v našem konkrétním případě modem telefonní, určený pro komutované telefonní okruhy.

.PI OBR08_1.TIF, 20, 40, 10 .PI OBR08_2.TIF, 20, 40, 10

Zapojíme-li na oba konce analogového okruhu (obecně telekomunikačního okruhu, viz obr. 8.1.) vhodné modemy, vznikne tím okruh umožňující přenos (číslicových) dat, a tudíž označovaný jako tzv.

datový okruh - data circuit,

Spojové organizace, které jsou vlastníky telekomunikačních okruhů, kladou přísné požadavky na zařízení UZD, která je možno k jejich okruhům připojit. V každém případě musí být toto zařízení spojovou organizací po technické stránce schváleno, tzv. **homologováno**. Některé spojové organizace dokonce na základě svého monopolu vyžadují, aby i zařízení UZD bylo jejich majetkem, a uživateli bylo pouze pronajímáno.

K datovému okruhu (přesněji na jeho ukončující zařízení) se pak již připojují taková zařízení, která jsou zdrojem či příjemcem (číslicových) dat. Tedy nejrůznější terminály a počítače.

V nepříliš intuitivní odborné terminologii se tato zařízení souhrně označují jako

koncová zařízení přenosu dat (KZD) - data terminal equipment (DTE).

Datový okruh, osazený na obou koncích zařízeními KZD, pak vytváří

datový spoj - data link.

Nejrůznější terminály či počítače se však nemusí připojovat přímo na ukončující zařízení datového okruhu (zařízení UZD). Může zde existovat ještě mezistupeň - např. koncentrátor terminálů, který dovoluje více jednotlivým terminálům sdílet jediný datový okruh. Jiný příkladem mohou být různé druhy komunikačních řadičů, které se zapojují mezi počítač a zařízení UZD, a přebírají na sebe většinu činností spojených s používáním datového okruhu, které by jinak musel vykonávat počítač.

9/ Veřejná telefonní síť

Veřejná telefonní síť je pro počítačové sítě velmi atraktivní. Ne snad proto, že by byla tím optimálním přenosovým kanálem pro přenos dat. Důvod je mnohem prozaičtější: veřejná telefonní síť existuje, je dostupná (téměř) odkudkoli a umožňuje snadno vytvořit dočasné spojení s kterýmkoli jiným místem ve svém dosahu. Zastavme se u ní proto poněkud podrobněji.

Běžný telefonní přístroj, v terminologii našich spojů "účastnické zařízení", je připojen na tzv.

účastnický přípojný okruh, účastnickou přípojku -

local loop, subscriber loop

který tvoří téměř vždy dva vodiče, vedoucí do nejbližší

(místní) telefonní ústředny - central office (end office, exchange office).

Veřejná telefonní síť je obvykle řešena hierarchicky a místní telefonní ústředny jsou pouze nejnižším článkem této hierarchie. Vyššími články jsou pak

uzlové telefonní ústředny - tandem offices, a

tranzitní telefonní ústředny - toll offices

Jednotlivé telefonní ústředny jsou mezi sebou propojeny pomocí tzv.

spojovacích vedení - trunks, interexchange channels (IXC).

Zařízení, které v rámci telefonní ústředny zajišťuje přepojování (komutaci) telefonních okruhů, se v angličtině označuje jako

switch, switchboard.

Vlastní přepojovací zařízení si mohou instalovat i různé uživatelské organizace pro svou vlastní neveřejnou telefonní síť. Pak se jedná o tzv.

pobočkovou ústřednu - private branch exchange (PBX),

která bývá obvykle napojena i na veřejnou telefonní ústřednu, takže kromě spojování hovorů mezi telefonními stanicemi v rámci příslušné organizace umožnuje i spojení s účastníky veřejné telefonní sítě. V angličtině se pro pobočkové ústředny používá také označení

private automatic branch exchange (PABX) resp.

computerized branch exchange (CBX),

je-li příslušné přepojovací zařízení pobočkové ústředny plně automatizováno resp. realizováno pomocí počítačové techniky.

Chceme-li se někomu telefonem dovolat, musíme nejprve zadat (vytočit) jeho telefonní (tzv. účastnické) číslo. Tím vlastně poskytujeme automatické telefonní ústředně informaci potřebnou k tzv. sestavení telefonního okruhu až k požadovanému účastníkovi.

Pro zadání telefonního čísla, tedy tzv.

volbu - dialing,

existují dva odlišné mechanismy. U nás se používá tzv.

pulsní volba - pulse dialing,

při které jsou jednotlivé číslice účastnického čísla vyjádřeny počtem impulsů, přenášených do telefonní ústředny po účastnickém přípojném okruhu. Tento mechanismus původně vznikl pro telefonní přístroje, vybavené rotační číselnicí, u kterých je počet impulsů dán dobou, potřebnou k otočení číselnice po jejím vychýlení zpět do výchozí polohy. To ovšem znamená, že vytočení různých telefonních čísel trvá různě dlouho. Snad i to byl důvod, proč např. v USA zavedli tzv.

tónovou volbu - tone dialing,

při které jsou jednotlivé číslice vyjádřeny nikoli počtem impulsů, ale tónem (tj. frekvencí) impulsu.

Zastavme se na chvíli právě u veřejné telefonní sítě v USA. Ta se totiž liší od evropských telefonních sítí i v dalších aspektech.

V Evropě i ve většině dalších zemí po celém světě provozují veřejné telefonní sítě spojové organizace (správy spojů), které patří státu, jsou jím přímo řízeny a na svou činnost mají přímo ze zákona stanoven monopol.

V USA jsou vlastníky a provozovateli spojových prostředků (včetně veřejné telefonní sítě) různé výdělečné organizace, které poskytují spojové služby na komerčním základě. Podstatné přitom je, že tyto organizace, označované jako

common carriers,

nejsou státními institucemi, ale organizacemi soukromého sektoru. Stát si pouze zachovává právo regulace a technického dozoru nad těmito organizacemi, a určuje maximální výše **tarifů** (poplatků za poskytované spojové služby), které si tyto organizace mohou účtovat. Za tímto účelem byl zřízen jeden orgán s celofederální působností,

Federal Communications Commission (FCC),

do jehož kompetence spadají komunikace z/do zahraničí a komunikace mezi jednotlivými státy v rámci USA, a v každém z těchto států pak orgán nazývaný

Public Utilities Commission (PUC),

který má místní působnost.

Poskytovatelů spojových služeb (common carriers) je v současné době v USA kolem 1200. Mezi největší patří firmy AT&T, sedm společností Bell Operating Companies (BOC), MCI Communications, US Sprint a General Telephone and Electronics.

Existence více poskytovatelů spojových služeb znamená jejich vzájemnou konkurenci, a v důsledku toho i existenci různých tarifů za spojení mezi stejnými místy. Zřizovatelé a provozovatelé počítačových sítí si tak mohou vybírat, od koho a za jakou cenu si objednají potřebné spojové služby, ať již jde o pevné okruhy či komutované okruhy veřejné telefonní sítě.

10/ Telefonní modemy

Jak jsme si již uvedli, běžný komutovaný telefonní okruh je řešen tak, aby byl schopen přenášet frekvence v rozsahu od 300 do 3400 Hz, což je pro běžné hovory telefonních účastníků vcelku dostatečné. Jak jsme již také dříve naznačili, šířka pásma 3100 Hz tohoto telefonního okruhu by měla umožňovat dosáhnout přenosových rychlostí teoreticky až kolem 30 000 bitů za sekundu. Prakticky dosahované rychlosti jsou ovšem mnohem nižší.

Abychom mohli přenášet data po telefonních okruzích, ať již pevných nebo komutovaných, tedy abychom z telefonního okruhu vytvořili okruh datový, potřebujeme modem.

Existuje celá řada telefonních modemů nejrůznějších typů a nejrůznějších vlastností. Každý z nich je však vždy konstruován pro určitou maximální přenosovou rychlost - modemy pro přenosové rychlosti 300 bitů/sekundu (bps, bits per second) jsou dnes již prakticky minulostí, rychle zastarávají i modemy pro 1200 bps. Na komutovaných okruzích se dnes používají nejčastěji modemy pro přenosovou rychlost 2400 bps. Stále více se však začínají prosazovat i modemy pro 9600 či 14400 bitů/sekundu, určené taktéž pro komutované okruhy, které dnes představují technologickou špičku. Pro pevné okruhy se používají speciální modemy pro takové přenosové rychlosti, které lze na příslušném pevném okruhu dosáhnout.

Samozřejmým požadavkem na každý modem je to, aby se dokázal "domluvit" s druhým modemem, který je zapojen na opačném konci telefonního okruhu. To znamená, že oba modemy musí používat stejný způsob modulace, stejnou přenosovou a modulační rychlost atd. - přičemž nemusí nutně jít o modemy stejného typu či modemy od stejných výrobců. Musí tedy existovat společný standard, který potřebnou kompatibilitu modemů umožní. Takovýmito standardy jsou v oblasti modemů doporučení mezinárodní organizace CCITT (o které si ještě povíme podrobněji). Tak například vlastnosti modemů pro komutované okruhy, umožňující plně duplexní provoz přenosovou rychlostí 2400 bps, určuje doporučení V.22 bis, obdobné doporučení pro modemy 9600 bps má označení V.32. Pro nižší přenosové rychlosti však existují i konkurenční standardy - např. pro rychlosti 1200 bps doporučení CCITT V.22 a americká norma Bell 212A, které nejsou vzájemně slučitelné.

Dnes vyráběné telefonní modemy mají obvykle mnoho užitečných vlastností a schopností. Většina z nich je např. schopna po dohodě s modemem na opačném konci okruhu přejít na nižší přenosovou rychlost v případě, že telefonní okruh je nekvalitní (má např. příliš mnoho poruch) a přenos původní rychlostí není únosný.

Starší modemy obvykle nebyly schopné si samy vytočit požadované telefonní (účastnické) číslo. K tomuto účelu musel buďto být použit běžný telefonní přístroj (který pak předal telefonní okruh modemu), nebo speciální "vytáčecí" zařízení, obvykle společné více modemům. Dnešní modemy si již umí vytočit potřebné telefonní číslo samy, tj. mají schopnost

automatické volby - auto dialing.

Jsou tedy schopné samy iniciovat a navázat spojení po komutované telefonní síti - pracují-li v tzv.

režimu volání - originate mode,

ale umí také samy "zvednout telefon", tj. odpovědět na přicházející volání, jsou-li v tzv.

režimu odpovědi - answer mode.

Díky tomu je možné, aby modemy navazovaly spojení a přenášely data po veřejné komutované síti i bez přímé lidské obsluhy resp. účasti. Každý modem však musí být možné vhodným způsobem ovládat, např. přinutit jej vytočit požadované telefonní číslo.

Dříve se modemy ovládaly pomocí různých řídících signálů, přepínačů a propojek. To však bylo příliš komplikované a nepružné. Různí výrobci modemů proto přicházeli s vlastními mechanismy ovládání - např. firma Hayes u svého příznačně pojmenovaného modemu Smartmodem (doslava "chytrý" modem) zavedla koncepci "plně programovatelného" modemu. Veškeré jeho ovládání se zajišťuje prostřednictvím řídících příkazů, které se modemu zasílají naprosto stejným způsobem, jako vlastní data, určená k přenosu. Tyto příkazy jsou navíc ve znakovém tvaru, takže jsou v případě potřeby srozumitelné i pro člověka (má-li k ruce jejich tabulku). Například příkaz pro vytočení telefonního čísla 123456 s použitím pulsní volby se zadá příkazem

ATDP123456

kde AT je návěští (uvozující každý příkaz modemu), D (od Dial) požaduje vytočení (volbu) čísla a P (od Pulse) specifikuje pulsní volbu. Modem na přijímané příkazy odpovídá opět znakově - např. úspěšné provedení příkazu potvrdí lakonickým "OK".

Způsob ovládání, zavedený firmou Hayes, se ukázal být natolik pružný a efektivní, že jej převzali i ostatní výrobci modemů a stal se (zatím nepsaným, ale dodržovaným) standardem, označovaným jako

Hayes standard.

Modemy, jejichž ovládání odpovídá tomuto standardu, se označují za Hayes-kompatibilní. Jsou to prakticky všechny dnes vyráběné modemy.

Příkazový jazyk, pomocí kterého se ovládají Hayes-kompatibilní modemy, se označuje také jako

(řídící) jazyk AT - AT (command) language

kvůli charakteristickému prefixu, který každý příkaz uvozuje.

Pomocí řídícího jazyka AT je možné nejen přímo řídit bezprostřední činnost modemu, ale je také možné nastavovat nejrůznější jeho parametry (kterých bývá nemálo) - např. kolikrát má v režimu odpovědi nechat "zazvonit telefon", než jej "zvedne". Většina dnešních modemů je vybavena nevolatilní pamětí pro čtení i zápis (tj. takovou, do které lze zapisovat, a která si svůj obsah podrží i po dobu, kdy není napájena). Díky tomu si modemy dokáží "pamatovat" své nastavení i po svém vypnutí.

11/ Využití veřejné telefonní sítě

Veřejná telefonní síť má pro komunikace s využitím počítačů a pro počítačové sítě obrovský význam. Vlastníte-li počítač, vhodný modem (který splňuje všechny požadavky příslušné správy spojů) a také komunikační program, schopný pracovat s tímto modemem, okamžitě se vám otevírá celá řada možností.

Můžete se např. domluvit s jiným šťastným vlastníkem podobné sestavy "počítač-modem-komunikační program", že mu v určenou dobu zavoláte. Vlastně že váš modem, ovládaný vaším komunikačním programem (ještě o nich bude řeč podrobněji) sám vytočí jeho telefonní číslo, a na něm se místo vašeho kolegy ozve opět modem, připojený k jeho počítači. Jakmile oba modemy naváží spojení, vstoupí do hry komunikační programy, běžící na obou počítačích. Jejich prostřednictvím pak můžete komunikovat s tím, kdo druhý počítač obsluhuje. Vaše komunikační programy vám obvykle nabídnou alespoň možnost **přenosu souborů (file transfer),** a možnost vzájemně si popovídat prostřednictvím zpráv, které vy píšete na svém počítači, a které se současně objevují i na monitoru vašeho protějšku - v angličtině se tomuto způsobu počítačového rozhovoru říká **chat.**

U protějšího, tzv. **vzdáleného počítače (remote computer)**, však nemusí nutně sedět člověk. Komunikační program, který právě běží na vzdáleném počítači, může být řešen tak, aby se s vámi dokázal dorozumět sám, i bez lidské obsluhy. Smyslem může být například to, abyste vy ze svého počítače, u kterého právě sedíte, mohli na dálku ovládat vzdálený počítač - prohlížet si jeho diskové adresáře, spouštět přenos souborů v kterémkoli z obou možných směrů apod., a to vše bez lidské účasti na "protější" straně. Pak jde o tzv. **dálkové ovládání (remote control)**, které jistě docení každý, kdo si kdy zapoměl při odchodu do práce zkopírovat ze svého domácího počítače na disketu důležitý soubor. Možnost "pohrabat" se na dálku (např. z počítače v kanceláři) ve svém domácím počítači prostřednictvím modemů a veřejné telefonní sítě pak může být skutečně k nezaplacení.

Ovládání počítače na dálku však skýtá ještě celou řadu dalších možností. Například různé poradenské a konzultační firmy v počítačově vyspělých zemích dnes instalují na počítačích svých zákazníků vhodný modem a komunikační program pro dálkové ovládání. V případě jakýchkoli problémů pak stačí, když pracovník poradenské firmy ze svého počítače "zavolá" počítač zákazníka. Program, který běží současně na obou počítačích, pak umožní, aby poradce na dálku pracoval s počítačem zákazníka, "viděl" jeho obrazovku, mohl na dálku "mačkat" klávesy na jeho klávesnici - zkrátka prováděl vše co je třeba, aby identifikoval příčinu zákazníkových problémů, a také mu mohl na dálku názorně předvést, co a jak má správně dělat.

Prostřednictvím veřejné telefonní sítě můžete také získat přístup k nejrůznějším informačním zdrojům a databázím, které mají povahu tzv. **on-line služeb (on-line services).** Jde opět o samostatné počítače vybavené modemem a vhodným komunikačním programem, které čekají, až je "zavolá" jiný počítač. Pracují v bezobslužném režimu, a uživateli volajícího počítače nabízí nejrůznější služby, od elektronické pošty přes poskytování informací např. o počasí, hodnotách akcií na burzách či letových řádech leteckých společností, až po databáze nejrůznějších právních předpisů a norem či výtahy článků známých časopisů. Navíc je možné se k těmto službám dostat skutečně odkudkoli, z libovolného místa v dosahu veřejné telefonní sítě. Samotné seznamy těchto on-line služeb vychází jako dosti obsažné knihy.

Přívlastek "on-line" (česky "spřažené") mají tyto služby z toho důvodu, že jejich uživatel je na ně se svým počítačem napojen přímo (prostřednictvím modemu a veřejné telefonní sítě), a bezprostředně komunikuje s programem na vzdáleném počítači, který příslušné služby zprostředkovává. Jde vesměs o interaktivní způsob vzájemné komunikace, při kterém on-line služba předkládá nabídku možných akcí, ze které si uživatel vybírá, a on-line služba pak reaguje podle výsledku jeho volby. Snad nejpopulárnější formou on-line služeb jsou dnes tzv. **Bulletin Board Systems (BBS)**, kterým stojí za to věnovat samostatný díl našeho seriálu.

Další možností využití veřejné telefonní sítě je připojení tzv. **vzdálených terminálů** (**remote terminals**) k "větším" počítačům. Zde si lze představit, že jednoúčelový terminál (typicky v sestavě monitor a klávesnice), vybavený modemem, "zavolá" velký počítač (obvykle tzv. **střediskový** resp. **sálový** počítač, anglicky **mainframe)**. Jakmile oba modemy naváží spojení, můžete ze vzdáleného terminálu pracovat s "velkým" počítačem obdobně, jako z tzv. **místního terminálu** (**local terminal**), který je ve fyzické blízkosti počítače.

Připojení vzdáleného terminálu prostřednictvím veřejné telefonní sítě je samozřejmě vhodné tam, kde je terminál využíván spíše příležitostně a jeho připojení pomocí pevného okruhu by bylo neekonomické.

Vzdálený terminál však nemusí nutně být skutečným terminálem (tzv. **dedikovaným terminálem (dedicated terminal)**), tedy jednoúčelovým zařízením, schopným plnit pouze úlohu terminálu. Může jít také o "plnohodnotný" počítač, např. počítač typu PC, který se díky vhodnému komunikačnímu programu pouze chová jako jednoúčelový terminál - tzv. jej **emuluje**, resp. vystupuje jako **emulovaný terminál (emulated terminal).** Je-li vybaven modemem, může se prostřednictvím veřejné telefonní sítě "dovolat" na velký počítač a po dobu existence spojení vystupovat v roli vzdáleného terminálu tohoto počítače.

Nesmíme však zapomínat ani na počítačové sítě v pravém slova smyslu. I pro ně má veřejná telefonní síť velký význam. Není totiž žádným problémem, aby si prostřednictvím veřejné telefonní sítě kdykoli "zavolaly" dva uzlové počítače a v bezobslužném provozu si navzájem předaly veškerá data, která si předat potřebují. Je to sice pomalejší a méně spolehlivé než kdyby mezi oběma uzlovými počítači existoval pevný okruh, umožňující trvalé spojení, ale na druhé straně to má i své velké výhody. Pevný okruh může být dosti nákladnou záležitostí, a není-li dostatečně využit, může jít o značně neefektivní investici. Také jeho pouhé zřízení může trvat dosti dlouho. Naproti tomu spojení po komutovaných linkách veřejné telefonní sítě je

okamžitě k dispozici, a náklady na něj jsou přímo úměrné provozu. Záleží vždy na charakteru konkrétní počítačové sítě, zda vyžaduje trvalé spojení mezi uzlovými počítači, velkou přenosovou rychlost a dostatečnou spolehlivost, což lze splnit pouze pomocí pevného okruhu, nebo zda vystačí se spojením prostřednictvím veřejné telefonní sítě, které je nutně pomalejší, méně spolehlivé a má dávkový charakter. Počítačové sítě, poskytující služby interaktivní povahy, nutně vyžadují trvalé spojení pomocí pevných okruhů. Naopak sítě, které nabízí převážně jen elektronickou poštu a přenos souborů, tedy služby spíše dávkové povahy, mohou vystačit se spojením pomocí komutovaných linek veřejné telefonní sítě. Příkladem mohou být sítě Eunet a Fidonet, o kterých si budeme povídat později.

12/ Bulletin Board Systems - BBS

Snad nejoblíbenější formou on-line služeb, dostupných prostřednictvím veřejné telefonní sítě, jsou dnes tzv. Bulletin Board Systems, zkráceně BBS. Místo pokusu o český překlad si raději vysvětleme, o co se jedná.

Bulletin Board System resp. BBS není nic jiného než počítač s jedním nebo několika modemy, který pracuje v bezobslužném režimu a čeká, až jej prostřednictvím veřejné telefonní sítě někdo zavolá. Vlastní funkce stanice BBS vykonává program, který na tomto počítači běží - jakmile se některý uživatel ze svého počítače na stanici BBS "dovolá", začne s tímto programem přímo komunikovat, a může využívat služby, které stanice BBS nabízí.

První stanice BBS se začaly používat zhruba před 15 lety, a sloužily výlučně pro předávání zpráv mezi svými uživateli. Kdokoli, kdo měl počítač vybavený vhodným modemem a znal potřebné telefonní číslo, se mohl na stanici BBS dovolat a zanechat zde svou zprávu či vzkaz, který si pak mohli ostatní uživatelé přečíst. Stanice BBS tak fungovala jako jakási nástěnka (anglicky: **bulletin board),** což dalo vzniknout jejímu názvu.

Schopnosti stanic BBS s časem rostly, a s nimi se samozřejmě vyvíjel i repertoár poskytovaných služeb.

Dnes je snad nejpopulárnější službou stanic BBS distribuce nejrůznějších programů. Jelikož dovolat se na stanici BBS může v podstatě každý, musí jít o programy, které lze legálně rozšiřovat, tedy programy z kategorie **public domain**, které lze používat a šířit zcela volně, nebo programy z kategorie **freeware**, které lze také volně šířit, ale ke kterým si jejich autoři udržují svá autorská práva, a uživatele vybízí k poskytnutí dobrovolného příspěvku na jejich další vývoj. Dále může jít o programy z kategorie **shareware**, které si zájemce může sám nejprve vyzkoušet, a shledá-li je účelnými a chce-li je využívat trvale, je vyzýván, aby se zaplacením předepsaného poplatku registroval u jejich autora. S komerčními programy, tedy těmi, které se běžně prodávají, bychom se na stanicích BBS neměli setkat. K dispozici však na stanicích BBS bývají i jiné druhy souborů, např. textové soubory s různou dokumentací, zdrojové tvary některých programů apod.

Programy resp. soubory, které stanice BBS nabízí, si uživatel může přenést do svého počítače postupem, který se označuje jako **download**. Existuje i opačná možnost, tedy přenos souborů od uživatele na stanici BBS, označovaný jako **upload**. I ten má svůj význam - stanice BBS tak mohou přijímat soubory od svých uživatelů, kteří je chtějí jejich prostřednictvím dále šířit mezi ostatními uživateli.

O každou stanici BBS se někdo musí starat - udržovat určitý řád v nabídce souborů, ošetřovat různé nestandardní situace (výpadky apod.) a zajišťovat mnohé další činnosti, nezbytné k provozu stanice BBS. Každá stanice má pro tento účel svého systémového operátora, který bývá označován zkráceně jako **sysop.**

Další služba, kterou stanice BBS obvykle poskytují, spadá do oblasti elektronické pošty. Uživatel může na stanici BBS poslat text ve formě **zprávy (message)**, která může být adresována buď některému jinému uživateli, nebo všem uživatelům. Odesilatel má možnost prohlásit odesílanou zprávu za soukromou - pak si ji může přečíst jen její adresát. Pokud tak neučiní, mohou si je přečíst a reagovat na ně i ostatní uživatelé. Jeden uživatel tak může formou zaslané zprávy "veřejně" vyslovit svůj názor na určitou věc, další uživatelé na to reagují svými názory a náměty ve formě zpráv, ještě další se připojují - a tak vznikají diskuse na určitá témata, kterým se v terminologii stanic BBS říká **konference.** Mohou sloužit nejen jako diskusní fórum, ale i jako zdroj nových kontaktů a informací - jednotlivé zprávy resp. příspěvky do konferencí mohou samozřejmě mít formu různých dotazů či žádostí o pomoc, zprostředkování kontaktu apod.

Konference mohou být lokální, tedy odehrávat se jen na dané stanici, nebo se do nich může zapojit více stanic BBS najednou. Ty si pak jednotlivé příspěvky samy automaticky předávají mezi sebou jako zvláštní druh pošty, tzv. **echomail**. Každý uživatel se tak může zapojit do diskuse z kterékoli stanice BBS, která se konference účastní, a má na této stanici k dispozici všechny příspěvky, které byly do konference zaslány.

Možnost vzájemného předávání příspěvků mezi stanicemi BBS dává tušit, že tyto bývají nějakým způsobem vzájemně propojeny. Vesměs opět prostřednictvím veřejné telefonní sítě - existuje dokonce všeobecně dodržovaný standard takovéhoto propojování stanic BBS do jednotné sítě **FIDONET**, která má dnes již celosvětový dosah.

Každá stanice BBS, která je zapojena do sítě FIDONET, může svým uživatelům zprostředkovávat "soukromou" elektronickou poštu, nazývanou **netmail.** Jejím prostřednictvím mohou uživatelé stanic BBS komunikovat s uživateli jiných stanic BBS prakticky po celém světě. Existují také tzv. **brány (gateways)** mezi sítí FIDONET a dalšími počítačovými sítěmi (např. sítí Internet), které umožňují přenos elektronické pošty do a z dalších počítačových sítí.

Stanice BBS jsou často provozovány počítačovými nadšenci na nevýdělečném základě, jsou přístupné každému a za své využití si neúčtují nic, nebo jen nevelký poplatek na krytí vlastních nákladů. Existují však i mnohé stanice BBS, které jsou provozovány různými firmami, organizacemi či institucemi, a mohou plnit také různé reklamní či jiné specifické funkce. Každá stanice BBS totiž kromě zpráv a souborů obvykle nabízí i další informace v různých formách. Může tedy sloužit i jako velmi účinný způsob reklamy pro svého provozovatele, který jejím prostřednictvím může nabízet své výrobky, služby apod. Další možností využití je např. poskytování poradenských služeb. Softwarové firmy je dnes nabízí nejčastěji pomocí tzv. **horkých linek (hot lines)**, jejichž prostřednictvím se můžete dovolat k technikům příslušné firmy. Stále častěji však softwarové firmy kromě horkých linek zřizují i specializované stanice BBS jako doplňkovou formu poskytování poradenských služeb. Uživatelé firemních produktů se mohou jejich prostřednictvím obracet na

firmu se svými dotazy či problémy, a firma jim může nejen odpovídat, ale i poskytovat nejrůznější dodatečné informace, návody, dodatky, doporučení, zkušenosti jiných uživatelů, demonstrační verze svých nových výrobků apod.

13/ Faksimilní přenos

Dalším, dnes velmi rošířeným způsobem využití veřejné telefonní sítě, který jen zdánlivě nemá s počítači a počítačovými sítěmi mnoho společného, je faksimilní přenos, zkráceně faxový přenos či pouze fax. Jaká je vlastně jeho podstata?

Faksimilní resp. faxový přenos je v principu přenos grafiky. V takové formě, jaká je dnes nejrozšířenější, představuje přenos jakékoli předlohy (např. tištěného či ručně psaného dokumentu, obrázku apod.) v číslicové formě po analogových linkách veřejné komutované telefonní sítě, prostřednictvím specializovaných faxových přístrojů.

Při odesílání nějakého dokumentu prostřednictvím faxového přenosu je nejprve nutné převést předlohu (tj. odesílaný dokument, chápaný jako grafický obrázek) do číslicové formy. Postup, kterým se tak děje, se nazývá **optickým snímáním (optical scanning)**, a zařízení, které jej provádí, je tzv. **snímač (scanner)**, zabudovaný do faxového přístroje**.**

Pro názornost si lze představit, že optický snímač se při snímání "dívá" na předlohu přes hustou síť, takže místo souvislého rovinného obrázku "vidí" jen jednotlivé izolované body. Nejjednodušší optické snímače vyhodnocují každý bod buď to jako černý nebo jako bílý, zatímco dokonalejší snímače rozeznávají u jednotlivých bodů určitý počet odstínů šedi. Existují samozřejmě také barevné snímače, které jsou schopné rozeznávat i barevné odstíny. Dnes používaný faksimilní přenos však nepracuje ani s barvou, ani s odstíny šedi. Při snímání ve faxovém přístroji je tedy každý bod předlohy chápán buď to jako černý nebo jako bílý. K vyjádření jeho stavu pak stačí jeden jediný dvojkový bit.

Objem dat, který vznikne nasnímáním celé předlohy, je samozřejmě závislý na hustotě našeho pomyslného síta, přesněji na **rozlišení (resolution)**, se kterým snímač faxového přístroje pracuje**.** Toto rozlišení je dnes u faksimilních přenosů standardně 200 bodů na palec ve vodorovném směru, a 100 bodů na palec ve svislém směru. Jen pro představu: jedna stránka běžného formátu A4 má rozměry přibližně 8,5 x 11 palců, při rozlišení 200x100 bodů to představuje celkem 200x8,5x100x11 neboli 1870000 bodů. Je-li každý z nich vyjádřen jedním bitem, je to 1,87 Megabitů resp. cca 234 kilobytů. A to ještě dnešní faxové přístroje obvykle nabízí i tzv. **fine** (jemný) režim, při kterém je rozlišení ve svislém směru dvojnásobné, tedy 200 bodů na palec.

Přenést čtvrt megabytu by trvalo neúměrně dlouho, proto dnešní faxové přístroje používají zvláštní techniku **komprese dat (data compression)**, která dokáže jejich objem snížit až přibližně 8-krát. Tím se objem dat, reprezentující jednu stránku A4, dostává na hodnoty řádově desítek kilobyte.

Číslicová data, která vzniknou optickým nasnímáním předlohy, se pak přenáší po analogových linkách veřejné komutované telefonní sítě prostřednictvím modemů. Modemy, zabudované ve faxových přístrojích, obvykle zvládají přenos rychlostí 9600 bitů/sekundu - v takovém případě trvá přenos jedné stránky formátu A4 pod půl

minuty. Při horší kvalitě linek je skutečná přenosová rychlost nižší, a doba přenosu jedné stránky se úměrně s tím prodlužuje.

Faxový přístroj příjemce na druhé straně telefonní linky může po nezbytné **dekompresi** přijatých dat zpětně sestavit z jednotlivých bodů původní předlohu. Vzniká tak opět grafický obrázek, který vytiskne bodová tiskárna, zabudovaná do faxového přístroje.

Dnešní velkou oblibu a široké nasazení faxů lze vysvětlit především tím, že umožňují přenést (téměř) jakoukoli předlohu kamkoli v dosahu veřejné telefonní sítě, a to prakticky okamžitě a při únosných nákladech. Navíc obsluha faxových přístrojů je obvykle tak jednoduchá, že je dokáže ovládat skutečně každý. Faxové přístroje bývají velmi často kombinovány s běžnými telefonními přístroji, a obvykle nabízí i různé další užitečné funkce, jako je např. automatické podávání jednotlivých listů předlohy, možnost naprogramovat odeslání na určenou dobu apod.

Faksimilní přenos ve své dnešní podobě má však i své četné nedostatky. Faxové přístroje nižších a středních kategorií jsou vybavovány zabudovanou tiskárnou, která netiskne na běžný papír, ale pouze na speciální teplotně citlivý papír. Také samotná kvalita tisku nebývá nejlepší. Rozlišení 200x100 resp. 200x200 bodů na čtvereční palec je dostatečné např. pro běžnou obchodní korespondenci, ale pro různé technické výkresy či podrobná schemata již nikoli. Také neschopnost přenášet barvu či alespoň odstíny šedi může být často na závadu. Při praktickém používání faxu může být velmi nepříjemná i skutečnost, že přenášená data nejsou nijak zabezpečována proti chybám při přenosu. Nejrůznější poruchy, kterých bývá ve veřejné telefonní síti často více než by bylo únosné, se pak projevují zkreslením přijatého obrázku. Záleží samozřejmě na intenzitě těchto poruch a míře zkreslení, které vyvolají, zda přenesený obrázek bude ještě srozumitelný či nikoli.

Výhodou faksimilního přenosu je také jeho relativně dobrá standardizace. Drtivá většina dnešních faxových přístrojů pracuje podle doporučení mezinárodní organizace CCITT pro faksimilní přenos tzv. **skupiny III (Group III)** - což je právě ten druh faksimilního přenosu, který jsme si doposud popisovali. Novější standard skupiny IV (Group IV), který zatím ještě není tak rozšířený, již pracuje s rozlišením až 400x400 bodů na palec, předpokládá přenos po pevných okruzích (tedy již nikoli prostřednictvím veřejné komutované telefonní sítě), a přenos jedné stránky A4 zvládne přibližně za pět sekund.

14/ Faksimilní přenos a počítače

Jak jsme si již naznačili minule, umožňuje faksimilní (faxový) přenos přenášet prostřednictvím veřejné telefonní sítě psané či tištěné dokumenty, obrázky, obecně jakékoli grafické předlohy.

Představme si ale nyní, že jsme prostřednictvím faksimilního přenosu přijali od svého spolupracovníka stránku textu, kterou potřebujeme dále zpracovat, nejlépe pomocí svého oblíbeného textového editoru. Problém je nyní v tom, že to co jsme přijali, je ve skutečnosti obrázek s grafickým vyobrazením textu (kterým může být například i rukou psaný text), a nikoli textový soubor, který by mohl náš textový editor zpracovávat. Máme proto v podstatě dvě možnosti - celý text ručně přepsat, nebo využít prostředků pro tzv. **optické rozpoznávání znaků (optical character recognition - OCR).**

Člověk, který čte psaný text, vyhledává v předloze grafické tvary, kterým přisuzuje význam jednotlivých písmen, z nich si pak skládá celá slova, z nich věty atd. Při optickém rozpoznávání znaků tuto činnost člověka vykonává program - také on se snaží vyhledávat na obrázku takové grafické tvary, které mu "připomínají" písmena, ta skládá v celá slova, věty atd., a výsledný text produkuje ve formě textového souboru, který již je obecně zpracovatelný textovými editory. Ačkoli pro člověka je čtení spíše rutinní záležitostí, pro program je to neobyčejně těžká úloha. Dnes samozřejmě existují mnohé programové balíky pro optické rozpoznávání znaků (tzv. **OCR programy)**. Jsou použitelné jen na tištěné písmo, nikoli na rukopisy, a především jejich účinnost resp. spolehlivost nebývá zdaleka stoprocentní. Je značně závislá na mnoha faktorech, jako např. na rozlišení, se kterým byla předloha nasnímána, na natočení předlohy apod. Navíc jde o velmi drahé programy, obvykle zahraniční provenience, které mohou mít nepřekonatelné problémy s českou a slovenskou diakritikou. Pro texty je tedy faksimilní přenos výhodný spíše v případě, že text nepotřebujeme dále zpracovávat na počítači.

Faksimilní přenos však není vázán jen na jednoúčelové faxové přístroje. Je dobré si uvědomit, že každý faxový přístroj se vlastně skládá ze tří zařízení - optického snímače, modemu a tiskárny - realizovaných jako jediný kompaktní celek. Tiskárny, navíc vesměs kvalitnější než u běžných faxových přístrojů, jsou běžnými periferiemi dnešních osobních počítačů, a také optické snímače již přestávají být vzácností. To, co ještě zbývá, abychom mohli odesílat a přijímat faxy na počítači, je vlastně již jen takový modem, jaký se zabudovává do faxových přístrojů. V principu jde o obdobný modem, jaký se používá při "běžných" datových přenosech, pouze konkrétní technika a parametry přenosu se liší (vychází z jiné normy resp. doporučení pro faksimilní přenos). Navíc je zde ještě otázka komprese a dekomprese dat, představujících číslicový tvar přenášené předlohy - viz minule.

Pokud má být počítač používán v roli faxového přístroje, musí být vybaven speciálním přídavným zařízením, které je schopné fungovat jako modem pro faksimilní přenos. V případě osobních počítačů je toto zařízení téměř vždy konstrukčně provedeno jako karta resp. deska, která se instaluje do jedné z rozšiřujících pozic**,** tzv. **slotů (slots)**. Jde tedy o **faxovou kartu (fax card)**. Další funkce, spojené s přijímáním a odesíláním faxů, se pak již zajišťují programovými prostředky. Jde mj. také o kompresi a dekompresi dat resp. převod dat z/do takového tvaru, v jakém se pak skutečně přenášejí. Vedle faxové karty tedy musí být na počítači instalován i potřebný software, který tyto funkce zajišťuje, a současně i ovládá vlastní faxovou kartu.

Potřebujeme-li z počítače odeslat prostřednictvím faxového přenosu nějaký text, který máme k dispozici ve formě textového souboru, nepotřebujeme k tomu nutně mít optický snímač (scanner). Převést textový soubor do grafické podoby, přesněji vyrobit z textového souboru grafický soubor, není žádným principiálním problémem - může to zajistit přímo obslužný program, který ovládá faxovou kartu a zajišťuje odesílání. Podobně je tomu v případě, že potřebujeme odeslat obrázek, který máme k dispozici ve formě vhodného grafického souboru. Například takový obrázek, který jsme vytvořili pomocí našeho oblíbeného programu pro malování (např. Paintbrush, Corel Draw apod.) - zde má faxový software ještě lehčí práci než u textových souborů. Optický snímač vlastně potřebujeme jen v případě, že máme odeslat obrázek, který

máme jen na papíře. Ten musíme s pomocí snímače nasnímat, tím jej získáme ve formě grafického souboru, a ten by již naše faxová karta měla umět odeslat.

Schopnosti některých faxových karet zde mohou končit - existují totiž faxové karty, určené jen pro odesílání.

Faxové karty, schopné také přijímat, produkují přijaté faxy opět ve formě grafických souborů. Ty lze vytisknout, nebo je dále zpracovat (jako obrázky!!) vhodným grafickým programem. Jde-li o texty, je principiálně možné je převést do formy textového souboru pomocí rozpoznávání znaků (OCR). Je také možné je pouze uschovat pro pozdější zpracování, ale zde je dobré si uvědomit, že ve zkomprimovaném formátu představuje jediná stránka velikosti A4 typicky několik desítek kilobyte dat, a v nezhuštěném tvaru (ve formě grafického souboru) řádově stovky kilobyte!

Kvalitní faxový přístroj může být sám o sobě velmi drahý, např. je-li místo málo kvalitní zabudované tiskárny vybaven tiskárnou laserovou, díky které může přijaté faxy tisknout kvalitně na běžný kancelářský papír. Kvalitní faxový přístroj tak může být typickým příkladem poměrně nákladné periferie, kterou se vyplatí sdílet v lokální počítačové síti. V takové úpravě, aby mohl být v síti používán, pak plní funkci tzv. **faxového serveru**. Z každého uzlu sítě je možné jeho prostřednictvím odeslat textový či grafický soubor jako fax stejným způsobem, jako kdyby šlo o faxovou kartu, instalovanou přímo v uzlovém počítači. Podobně může faxový server sloužit i pro příjem faxů. Zde však nastává malý problém - kterému uzlu v síti má být přijatý fax předán? Faxy se odesílají vždy na určité telefonní číslo, kde je někdo prohlédne, a pak případně předá dál. Jak to ale dělat v případě faxového serveru, který slouží např. lokální síti se stovkami uzlových počítačů?

15/ PCM a spoje T

Když v roce 1837 představil Samuel Morse veřejnosti svůj telegraf, ve své podstatě digitální, předběhl tím o plných 39 let Alexandra Grahama Bella, který přišel se svým analogovým telefonem až v roce 1876. Analogový způsob přenosu však od počátku v oblasti telekomunikací zcela převládl, a teprve v historicky nedávné době se začal prosazovat přenos digitální. Svůj nemalý podíl na tom mají i číslicové počítače a počítačové sítě.

Přenos dat přímo v jejich číslicovém (digitálním) tvaru, tedy digitální přenos, má oproti přenosu analogovému četné výhody. Digitální přenosové cesty dnes umožňují dosahovat mnohem vyšších přenosových rychlostí, digitální signál se mnohem snáze zpracovává než signál analogový a příznivější jsou i ekonomické ukazatele. Není proto divu, že téměř všude se přechází na digitální přenos, a to například i v oblasti veřejných telefonních sítí. Moderní telefonní ústředny jsou dnes již výhradně digitální a jsou navzájem propojovány pomocí digitálních přenosových cest.

Jak ale přenášet analogový signál, např. telefonní hovor, po digitálních přenosových cestách? Zařízení, které převádí analogový telefonní signál do číslicového tvaru, se označuje jako **codec**, což je zkratka do **COder/DECoder.** Jde v podstatě o opak modemu, který na základě posloupnosti číslicových dat generuje spojitý analogový signál. Codec naopak na základě analogového signálu generuje posloupnost číslicových dat. Dělá to tak, že pravidelně vzorkuje stav analogového signálu a jeho okamžitou hodnotu aproximuje jednou ze 128 resp. 256 úrovní, které rozeznává - viz

obr. 15.1 - a kterou pak vyjádří jako jedno 7-bitové resp. 8-bitové číslo. Analogový telefonní signál se přitom považuje za signál o šířce pásma 4000 Hz, z čehož plyne, že jej stačí vzorkovat 8000-krát za sekundu (tj. každých 125 mikrosekund). Codec tedy generuje celkem 8000 x 7 bitů, tj. 56 kbitů za sekundu, resp. 8000 x 8 bitů, tj. 64 kbitů za sekundu. Pro přenos jednoho telefonního hovoru v digitální formě je pak nutná přenosová rychlost 56 resp. 64 kbit/sekundu.

Právě popsaná technika převodu analogového signálu do číslicového (digitálního) tvaru se označuje jako **impulsová kódová modulace (Pulse Code Modulation)**, zkratkou PCM. Je velmi rošířenou, nikoli však jedinou možnou technikou digitalizace - existují další, jako např. **adaptivní diferenciální impulsová kódová modulace (Adaptive Differential Pulse Code Modulation, ADCPM)** nebo tzv. **delta modulace**, které pro přenos hlasového kanálu v digitální formě vystačí s menší přenosovou rychlostí.

V době, kdy se digitální přenos analogových signálů začal stávat reálně použitelnou technologií, byl nejvyšší čas vypracovat pro ni jednotný standard. Členové mezinárodní organizace CCITT, která je k tomu příslušná, však nebyli schopni se včas dohodnout, a tak vzniklo několik navzájem nekompatibilních standardů.

.PI OBR15_1.TIF, 20, 40, 10

Obr. 15.1.: Představa impulsové kódové modulace (PCM)

V USA se digitální přenosové cesty používaly již od šedesátých let. Společnost AT&T, která v té době díky svému monopolnímu postavení provozovala přes 90% všech telefonních sítí v USA, je používala k propojování svých telefonních ústředen. Později však začala tyto spoje, označované jako **spoje T** (**T Circuits)**, pronajímat také jiným organizacím.

V dnešní době existuje celá hierarchie těchto spojů, odstupňovaných podle přenosových rychlostí - viz tabulka 15.2. Jedná se o dvoubodové digitální přenosové cesty, které lze používat buď jako jediný digitální kanál, nebo je rozdělit na několik podkanálů. Typicky se používají pro přenos telefonních hovorů v digitální formě. V takovémto případě jsou rozděleny na podkanály s přenosovou rychlostí 64 kbit/sekundu, a analogový telefonní signál je pro potřeby přenosu kódován pomocí impulsové kódové modulace. Počet telefonních hovorů, které lze po jednotlivých variantách T spojů přenášet, uvádí tabulka 15.2.

Firma AT&T vytvořila také celou hierarchii standardů, označovaných jako **North American Digital Hierarchy**, které definují způsob přenosu hlasových kanálů po jednotlivých variantách spojů T - viz tabulka 15.3.

T spoj Počet hlasových kanálů Přenosová rychlost

(při kódování PCM) (bitů/sekundu)

T1 24 1544000

T1C 48 3152000

T2 96 6312000

T3 672 44736000

T4 4032 274176000

Tabulka 15.2.: Spoje T

Formát Počet Ekvivalentní Přenosová rychlost
hlasových počet spojů T1 (bitů/sekundu)
kanálů

DS-0 1 - 64000

DS-1 24 1 154400

DS-1C 48 2 3152000

DS-2 96 4 6312000

DS-3 672 28 44736000

DS-3C 1344 56 90631000

DS-4E 2016 84 139264000

DS-4 4032 168 274176000

Tabulka 15.3.: North American Digital Hierarchy

Ukažme si podrobněji, jak to vypadá v případě spojů **T1**, které jsou stále ještě nejrozšířenější. Formát přenášených dat definuje pro tento typ spojů T standard **DS-1.** Předpokládá, že 24 hlasových kanálů je střídavě přepínáno (multiplexováno) na vstup jediného zařízení codec, které vzorkuje hodnotu analogového signálu a vyjadřuje ji

jako 7-bitové číslo. K těmto sedmi datovým bitům se pak přidává ještě jeden řídící bit, takže každý hlasový kanál přispívá při každém svém vzorkování do přenášených dat celkem osmi bity. Vzhledem k pravidelnému přepínání mezi jednotlivými hlasovými kanály jsou tyto 8-bitové položky od jednotlivých kanálů řazeny za sebe - 24 osmibitových položek od jednotlivých hlasových kanálů, doplněných ještě jedním řídícím bitem, pak tvoří tzv. **rámec (frame)**. Ten má celkem (24 * 8) + 1 = 193 bitů, viz obr. 15.4. Jednotlivé hlasové kanály se vzorkují 8000-krát za sekundu, což odpovídá 8000 rámců za sekundu. Má-li každý rámec 193 bitů, je zapotřebí přenosová rychlost 8000 * 193 = 1544000 resp. 1,544 Mbitů za sekundu, což je právě přenosová rychlost spojů T1.

.PI OBR15_4.TIF, 20, 40, 10

Obr. 15.4.: Přenosový rámec spojů T1

Uživatelé v USA si dnes mohou pronajímat spoje T v takové variantě, která odpovídá jejich potřebám přenosových rychlostí. Ne vždy však dokáží plně využít možností i té "nejpomalejší" varianty, spojů T1. Provozovatelé spojů T1 je dnes proto pronajímají již také po jednotlivých podkanálech (jako tzv. **fractional T1)**, nejčastěji pro 384, 512 nebo 768 kilobitů za sekundu, přičemž sami zajišťují potřebné dělení přenosového pásma (multiplexing).

Když organizace CCITT konečně dospěla k jednotnému standardu pro přenos hlasových kanálů po digitálních přenosových cestách, zdálo se jí, že jeden řídící bit na každých sedm datových bitů - jako je tomu u spojů T1 či přesněji u standardu DS-1 - je příliš velkou režií. Standard CCITT proto předpokládá kódování okamžité hodnoty analogového signálu do 8 bitů, místo do 7 bitů jako v případě standardu DS-1. Místo 24 kanálů jich předpokládá celkem 32, z toho 30 hlasových, a dva pomocné řídící. Vzorkování se provádí opět 8000-krát za sekundu, potřebná přenosová rychlost tudíž vychází: 8000 x 32 x 8 = 2048000, tj. 2,048 Mbitů za sekundu. Příslušný standard CCITT se správně jmenuje G703/732, ale často se označuje také jako **evropský T1 spoj** nebo též jako **spoj E1**. Existují také analogická doporučení CCITT pro spoje s přenosovými rychlostmi 8,848, 34,304, 139,264 a 565,148 Mbitů/sekundu.

16/ ISDN

Telefonní síť byla po více než sto let hlavní komunikační infrastrukturou, vytvořenou s ohledem na potřeby analogového přenosu lidského hlasu. Uživatelé však dnes požadují nejen vyšší kvalitu hlasových přenosů, než pro jaké je stávající telefonní síť koncipována, ale také začlenění dalších nových služeb. Jak by se vám líbil například telefon, který by kromě vyzvánění ukazoval také např. číslo toho účastníka, který vás právě volá? Nebo rychlejší a kvalitnější faksiimilní přenos (skupiny IV - viz 13. díl našeho seriálu), který zvládne přenos jedné stránky A4 do pěti sekund? Co takhle možnost normalizovaného připojení terminálu nebo počítače přímo na veřejnou přenosovu síť?

Jak jsme se již zmínili minule, spojové organizace samy z technických důvodů přechází na digitální telefonní ústředny a jejich vzájemné propojení pomocí digitálních přenosových cest. Výše naznačené požadavky uživatelů tento přechod o to více motivují, požadují jeho rozšíření až ke koncovým uživatelům, a přidávají ještě požadavek dalších nových služeb. Výsledkem je snaha o přeměnu celé nynější analogové telefonní sítě na síť digitální, která by integrovala jak hlasové, tak i další

přenosové služby. Nový jednotný systém, ke kterému by měl tento vývoj směřovat, se nazývá **ISDN (Integrated Services Digital Network resp. Digitální telekomunikační síť s integrovanými službami)**, a je koordinován mezinárodní organizací CCITT formou jejích doporučení.

Koncepce ISDN předpokládá, že bude svým uživatelům poskytovat různé druhy přenosových kanálů resp. okruhů. Existuje pro ně i jednotné označení:

A - analogový telefonní kanál s šířkou pásma 4 kHz,

B - digitální kanál s přenosovu rychlostí 64 kbit/sekundu pro přenos hlasu (s kódováním PCM) nebo dat,

C - digitální kanál s přenosovou rychlostí 8 nebo 16 kbit/sekundu

D - digitální kanál pro služební účely, s přenosovou rychlostí 16 nebo 64 kbit/sekundu,

E - digitální kanál s přenosovou rychlostí 64 kbit/sekundu pro interní potřeby ISDN

H - digitální kanál s přenosovou rychlostí 384, 1536 nebo 1920 kbit/sekundu.

Koncepce ISDN však nepředpokládá libovolnou skladbu těchto kanálů. Doporučení CCITT zatím standardizuje tři různé kombinace těchto kanálů, označované jako:

Basic rate (2 kanály B a 1 kanál D),

Primary rate (v USA a Japonsku: 23 kanálů B a 1 kanál D, v Evropě 30 kanálů B a 1 kanál D),

Hybrid (1 kanál A a 1 kanál C).

Smyslem poslední z uvedených kombinací (hybrid) je umožnit připojení stávajících analogových telefonů na síť ISDN. Jde však spíše o jakési nouzové řešení. Mnohem zajímavější jsou zbývající dvě možnosti. První z nich (Basic rate) je zamýšlena jako náhrada stávající účastnické telefonní přípojky do domácností, malých kanceláří nebo např. k pracovním stolům jednotlivých zaměstnanců v rámci větších organizací. Každý z obou kanálů B je schopen přenášet jeden telefonní hovor s kódováním PCM (viz minule), nebo data rychlostí 64 kbit/sekundu, přičemž pro veškeré řízení (např. "vytáčení" hovorů, přenos údajů o účastnickém čísle volajícího, o poplatcích za spojení apod.) slouží služební kanál D s přenosovou rychlostí 16 kbit/sekundu.

Myšlenka poskytnout účastníkovi dva datové přenosové kanály místo jednoho je sama o sobě zajímavá. V pozadí zřejmě stojí důvody snažšího prosazení a marketingu této nové technologie, která svými dvěma přenosovými kanály nabízí již na první pohled zřejmé vylepšení oproti stávajícímu stavu. Typickým způsobem využití má být jeden telefonní hovor mezi dvěma účastníky (po jednom z obou kanálů B), který může být doplněn např. přenosem dat mezi stejnými účastníky po druhém kanálu B. Představte si například, že se svým kolegou připravujete společný článek. Vaše počítače, na kterých dokument připravujete, jsou navzájem propojeny jedním datovým kanálem a oba vidíte přímo před sebou stejnou verzi dokumentu, do které se vám ihned promítají všechny průběžně prováděné změny. Díky prvnímu kanálu B se můžete se svým kolegou domlouvat vedle toho ještě i hlasem - co více si přát?

Dva kanály B a jeden kanál D, to je 2 * 64 + 16 = 144 kbitů za sekundu, s režií 48 kbit/sekundu na multiplex všech tří kanálů pak celkem 192 kbit/sekundu. Otázkou je,

kde vzít takto rychlou přenosovu cestu od koncového uživatele až k telefonní ústředně? Odpověď je jednoduchá - lze využít již existující účastnické přípojky pro stávající analogovou telefonní síť. Přibližně 80 procent účastnických přípojek je totiž kratších než 7 - 8 kilometrů, a lze na nich dosáhnout přenosové rychlosti až 2 Mbity/sekundu. Na těch zbývajících pak o něco méně.

Účastnická přípojka (účastnický přípojný okruh), vedoucí od koncového uživatele k telefonní ústředně, je sama o sobě jediným přenosovým okruhem, který je mezi kanály B a D rozdělen pomocí časového multiplexu (viz 5. díl našeho seriálu). Na konci této účastnické přípojky pak musí být umístěno ukončující zařízení, v terminologii ISDN označované jako **NT1 (Network Termination)**, viz obr. 16.1. Toto ukončující zařízení vytváří také standardní rozhraní ISDN, ke kterému lze připojit až 8 zařízení, vyhovujících stejnému standardu - tedy např. telefonní přístroj pro ISDN, ISDN terminál apod.

.PI OBR16_1.TIF, 20, 40, 10

Obr. 16.1.: Jednoduchá přípojka ISDN

.PI OBR16_2.TIF, 20, 40, 10

Obr. 16.2.: Přípojka ISDN s pobočkovou ústřednou

Pro ty uživatele, kterým nestačí ISDN přípojka typu Basic rate (např. pro organizace), je určena přípojka Primary rate s vyšším počtem datových kanálů B a služebním kanálem D s přenosovou rychlostí 64 kbit/sekundu. Předpokládá se, že koncový uživatel (organizace) si k této přípojce připojí vlastní digitální pobočkovou ústřednu pro ISDN, ke které si pak může připojovat větší počet různých zařízení se standardizovaným ISDN rozhraním, a přes vhodné adaptéry i zařízení s jiným rozhraním, např. RS-232-C. V terminologii ISDN je tato ústředna označována jako zařízení **NT2**, a připojuje se k ukončujícímu zařízeni NT1 - viz obr. 16.2.

Odlišnost v počtu kanálů B v USA a v Evropě je u přípojky Primary rate dána odlišným standardem pro spoje T1, které se pro tyto přípojky používají nejčastěji. 23 kanálů B a jeden kanál D představuje celkem 23 * 64 + 64 = 1536 kbit/sekundu, což se právě "vejde" na jeden spoj T1 dle severoamerického standardu, který má přenosovou rychlost 1,544 Mbit/sekundu (viz minule), zatímco evropský spoj T1 nabízí 2,048 Mbit/sekundu, a kromě jednoho kanálu D po něm lze tedy přenést ještě 31 kanálů B, z nichž je ovšem jeden vyhražen pro potřeby řízení a správy**.**

Základní doporučení k ISDN byla organizací CCITT vydána již v roce 1984, do praktického života se však tato nová technologie prosazuje relativně pomalu. Rozhodně pomaleji, než její zastánci původně předpokládali. Snad za to může i přetrvávající nedostatečná informovanost a nevyjasněnost některých koncepčních otázek. V době vzniku celé koncepce zlé jazyky tvrdily, že ISDN správně znamená "I Sure Don't kNow", tedy doslova "opravdu nevím", zatímco dnes to prý znamená "I Still Don't kNow", neboli "stále ještě nevím". Jiné prameny zase uvádí, že jde o zkratku: "I Sure Don't Need it", neboli "opravdu to nepotřebuji". Kdo ví.

17/ Veřejné datové sítě, přepojování okruhů a přepojování paketů

V rozlehlých počítačových sítích se uzlové počítače propojují po dvojicích mezi sebou pomocí tzv. **dvoubodových spojů (point-to-point lines)** - viz obr. 17.1.a/. Nikoli ovšem každý s každým, takže spojení mezi dvěma počítači může vést přes několik mezilehlých. V tomto případě si počítačová síť musí sama zajistit vše, co je třeba, aby se přenášená data dostala až do cílového uzlu: např. volbu nejvhodnější cesty (která nemusí být zdaleka jen jedna), přijetí dat v mezilehlém uzlu a jejich následné odeslání dále k cílovému uzlu atd.

.PI OBR17_1.TIF, 20, 40, 10

Obr. 17.1.: Propojení uzlů počítačové sítě

a/ pomocí dvoubodových spojů

b/ pomocí veřejné datové sítě

Spojové organizace (správy spojů) však mohou nabízet ještě jinou alternativu, při které na sebe přebírají všechny právě naznačené povinnosti, spojené s doručením dat až k jejich skutečnému adresátovi. Jako další službu veřejnosti mohou nabízet vlastní síť pro přenos dat, tzv. **veřejnou datovou síť (Public Data Network, PDN)**, na kterou si mohou uživatelé připojovat své počítače, terminály apod.

Z pohledu svých uživatelů se veřejná datová síť jeví jako jedna velká černá skříňka s určitým počtem přístupových bodů, ke kterým se pomocí dvoubodových spojů (přípojek) napojují uživatelské počítače - viz obr. 17.1.b/. Pro ně pak veřejná datová síť funguje jako jedno velké propojovací pole - dokáže přenést data z kteréhokoli svého přístupového bodu do kteréhokoli jiného.

Vnitřní struktura veřejné datové sítě není pro uživatele relevantní, a často se ani nezveřejňuje. Co však musí být velmi přesně definováno, zveřejněno a dodržováno, jsou přesná pravidla hry při připojování čehokoli na veřejnou datovou síť, tedy rozhraní mezi veřejnou datovou sítí a připojeným počítačem.

.PI OBR17_2.TIF, 20, 40, 10

Obr. 17.2.: Představa sítě s přepojováním okruhů

Pro vlastní transport dat může veřejná datová síť (tak jako obecně každá síť) používat různé mechanismy. Jednou z možností je tzv. **přepojování okruhů (circuit switching)**, které lze nejlépe přirovnat k mechanismu činnosti veřejné telefonní sítě. Také zde totiž dochází k propojování přenosových cest tak, aby mezi dvěma přístupovými body k veřejné datové síti vzniknul jediný přenosový okruh, v tomto případě datový okruh - viz obr. 17.2. Tento datový okruh musí být nejprve sestaven (na žádost toho, kdo spojení přes veřejnou datovou síť iniciuje) a pak existuje až do té doby, než je opětně rozpojen. Oba účastníci spojení tak mají po celou dobu své vzájemné komunikace k dispozici a výlučně pro sebe datový okruh, po kterém si mohou vyměňovat data v podstatě jakýmkoli způsobem. Mohou si posílat libovolně velké bloky dat, nebo také např. spojitý proud bytů.

Další možností je tzv. **přepojování paketů (packet switching)**. To je založeno na myšlence, že přenášená data se rozdělí na stejně velké bloky, a k nim se doplní některé další údaje (mj. adresa odesilatele a adresa příjemce). Tím vznikají tzv. **pakety (packets)**, které se pak přenáší datovou sítí jako dále nedělitelné celky. Nevzniká zde ovšem žádný skutečný datový okruh mezi odesilatelem a příjemcem

paketu. Místo toho si jednotlivé vnitřní uzly veřejné datové sítě předávají pakety mezi sebou, dokud je nedoručí až k přístupovému bodu, na který je napojen adresát paketu.

Mechanismus přepojování paketů může být realizován dvěma odlišnými způsoby. První z nich předpokládá, že každý vnitřní uzel datové sítě, který přijme nějaký paket, se vždy znovu rozhoduje o tom, kudy ho má poslat dál. Jde o analogii běžné listovní pošty, ve které datový paket odpovídá dopisní obálce, opatřené adresou příjemce, a vnitřní uzel datové sítě hraje roli poštovního úřadu, který musí rozhodnout o tom, kam obálku předá dál. Analogii s listovní poštou symbolizuje i název této varianty přepojování paketů, které se říká **datagramová služba**, a datovým paketům **datagramy (datagrams)**. Výhodou je možnost, aby datová síť při přenosu datagramů průběžně reagovala na své okamžité zatížení, a v případě potřeby volila různé alternativní cesty přenosu. V důsledku toho pak datová síť nezaručuje správné pořadí doručování jednotlivých datagramů - může se totiž stát, že díky použití alternativních tras uvnitř datové sítě některé datagramy "předbíhají" - viz obrázek 17.3.

.PI OBR17_3.TIF, 20, 40, 10

Obr. 17.3.: Představa sítě s datagramovou službou

Nevýhodou datagramové služby je režie, spojená s rozhodováním o dalším směru přenosu. Tu se snaží minimalizovat alternativní způsob realizace mechanismu přepojování paketů, označovaný jako mechanismus **virtuálních spojů (**též: **virtuálních spojení, virtuálních okruhů - virtual calls, virtual circuits)**. Aby se každý vnitřní uzel datové sítě nemusel pro každý nový paket znovu rozhodovat, kudy ho pošle dál, zvolí se ještě před vlastním přenosem dat jedna pevná cesta od odesilatele k příjemci, a po celou dobu přenosu se pak používá právě a pouze tato cesta - viz obr. 17.4. Celý mechanismus funguje tak, že nejprve se od odesilatele vyšle zvláštní paket, který "vytyčí" momentálně nejvhodnější cestu mezi oběma účastníky, a informace o ní zanese do každého vnitřního uzlu, který na této cestě leží. Vzniká tak analogie datového okruhu, který ovšem existuje pouze jako konvence o tom, kudy se mají pakety přenášet - proto přívlastek "virtuální". Vlastní pakety, které se odesílají virtuálním spojem, nejsou opatřovány adresou svého příjemce, ale označením virtuálního spoje, který k jejich příjemci vede. Každý vnitřní uzel datové sítě, který takový paket přijme, se pouze podívá do svých tabulek, kudy příslušné virtuální spoj pokračuje dál, a tím směrem přijatý paket opět předá.

.PI OBR17_4.TIF, 20, 40, 10

Obr. 17.4.: Představa sítě s virtuálními spoji

Virtuální spoje se tedy zřizují až v okamžiku potřeby přenosu dat, a po jeho dokončení se zase ruší. Existuje ovšem i jiná možnost, tzv.**pevné virtuální spoje (**též: **pevné virtuální okruhy - permanent virtual calls, permanent virtual circuits)**. V tomto případě jsou potřebné informace o "trase" virtuálního spoje uchovávány ve vnitřních uzlech datové sítě trvale. Tím odpadá potřeba zřizování a následného rušení virtuálního spoje, který tak existuje i v době, kdy není využíváno.

K veřejným datovým sítím, které používají mechanismus přepojování paketů (sítím **PSPDN, Packet Switched Public Data Networks**) lze samozřejmě připojovat jen

taková zařízení, která jsou schopna s pakety požadovaného formátu pracovat. Existuje ovšem i jiná možnost - svěřit sestavování dat do paketů a jejich zpětné "vybírání" specializovanému zařízení, které pak může sloužit potřebám více jednodušších koncových zařízení (např. terminálů), které s pakety pracovat neumějí. Jedná se o tzv. zařízení **PAD (Packet Assembler/Disassembler)**.

Veřejné datové sítě jsou dnes ve světě značně rozšířené, a to převážně v podobě sítí s přepojováním paketů a s virtuálními spoji. Jsou budovány podle doporučení X.25 organizace CCITT, které je dnes standardem pro tento druh veřejných datových sítí (a my si o něm budeme ještě povídat podrobněji).

Podle místních legislativních podmínek jsou veřejné datové sítě buď majetkem příslušné správy spojů, která má na jejich zřizování monopol, nebo mohou být provozovány i jinými organizacemi. Vždy však na komerčním základě. Uživatel platí za využití veřejné datové sítě buď paušálně, nebo podle objemu přenesených dat, nejčastěji však kombinací obou způsobů.

Pro provozovatele počítačových sítí je veřejná datová síť lákavou možností, jak propojit jednotlivé uzlové počítače své sítě. Minimalizuje jednorázové počáteční náklady na zřízení komunikační infrastruktury a průběžné náklady na její správu. Vzhledem k obvyklé závislosti poplatků za přenos na objemu dat však vykazuje vyšší provozní náklady. Záleží tedy vždy na konkrétní tarifní politice a objemu provozu v počítačové síti, kdy se ještě vyplatí využívat služeb veřejné datové sítě a kdy je výhodnější si zřídit vlastní komunikační infrastrukturu.

18. Mikrovlnné a družicové spoje

Rozlehlé počítačové sítě využívají k propojení svých uzlových počítačů nejčastěji pevné okruhy, pronajaté od spojových organizací. Tyto okruhy jsou obvykle vytvářeny pomocí drátových přenosových cest (různých dálkových kabelů apod.). Existují však i jiné možnosti (kromě veřejných datových sítí) - například mikrovlnné a satelitní spoje.

Přívlastkem **mikrovlnné (microwave)** se označují elektromagnetické vlny o extrémně krátké vlnové délce resp. velké frekvenci, která je vlnové délce nepřímo úměrná. V praxi se používají frekvence od 1 do 12 GHz (tj. vlnové délky přibližně 30 až 2,5 cm). Vlny o takto vysoké frekvenci již lze, pomocí vhodných parabolických vysílacích antén, soustředit do úzkého paprsku, a ten nasměrovat na přijímací anténu. Úzce soustředěný paprsek vykazuje minimální rozptyl, dovoluje používat relativně malý výkon vysílače a je velmi odolný vůči rušení. Na nižších frekvencích nelze dosáhnout potřebného soustředění, a na vyšších frekvencích se již začíná znatelně projevovat nepříznivý vliv atmosférických jevů, jako např. mlhy a deště.

.PI OBR18_1.TIF, 20, 40, 10

Obr. 18.1.: Mikrovlnné spoje

Vzhledem k přímočarému šíření soustředěného paprsku elektromagnetických vln je dosah mikrovlnných spojů omezen na přímou viditelnost vysílače a přijímače. Ta je určována konkrétními geografickými podmínkami, a samozřejmě také zakřivením Země. Lze ji uměle prodlužovat umisťováním vysílacích a přijímacích antén na co nejvyšší věže. V rovině, kde se uplatňuje pouze vliv zakřivení zemského povrchu, je

obvyklý dosah kolem 50 km. Pro překlenutí větších vzdáleností je nutné budovat síť retranslačních stanic - viz obr. 18.1.

Dosažitelná přenosová rychlost na mikrovlnných spojích je závislá na použitém frekvenčním pásmu a možnostech přijímače a vysílače. Může dosahovat hodnot až 10 Mbit/sekundu.

Pro počítačové sítě mohou být mikrovlnné spoje výhodné například v městských aglomeracích v těch místech, kde neexistují vhodné drátové přenosové cesty a instalace nových nepřipadá v úvahu (např. v historických jádrech měst).

Přímá viditelnost mezi přijímačem a vysílačem je samozřejmě značně omezujícím faktorem. Jednou z možností, jak se mu vyhnout, je nechat odrazit úzce směrovaný paprsek od horních vrstev troposféry (ve výšce cca 16 km) - viz obr. 18.2. Nevýhodou těchto tzv. **troposférických spojů** je nutnost používat velký výkon vysílače (desítky kW), přičemž přijímaný signál bývá velmi slabý - jen asi 1 nW.

.PI OBR18_2.TIF, 20, 40, 10

Obr. 18.2.: Troposferický spoj

S další možností - využitím družice - poprvé přišel známý spisovatel sci-fi, Arthur C. Clarke, a to již v roce 1945, tedy 12 let před vypuštěním prvního sputniku.

Dnes se pro **družicové spoje (satellite links)** používají převážně tzv. **geostacionární** družice, někdy označované též jako tzv. **synchronní** družice), které se pohybují ve výšce přibližně 36 000 km nad zemí. V této výšce je doba jejich obletu kolem Země rovna době otočení zeměkoule kolem její osy, což znamená, že se nacházejí vždy nad stejným bodem zemského povrchu - většinou se takto umisťují nad rovník.

Družice může fungovat jen jako pouhý odražeč signálu - pak jde o tzv. **pasivní družice.** Výhodnější jsou však tzv. **aktivní družice.** Ty obsahují vždy několik tzv. **transpondérů (transponders)**, které fungují jako na sobě nezávislé retranslační stanice. Přijímají signál vysílaný ze Země (tzv. **up-link signal**), převádí jej do jiného frekvenčního pásma, zesilují jej a vysílají zpět směrem k Zemi (jako tzv. **down-link signal**) - viz obr. 18.3.

.PI OBR18_3.TIF, 20, 40, 10

Obr. 18.3.: Družicový spoj

Pro družicové spoje se používají různá frekvenční pásma. V tz. **C-pásmu (C-band)** se ze Země k družici vysílá v pásmu 6 GHz, a v opačném směru v pásmu 4 GHz. Jsou k tomu zapotřebí relativně velké parabolické antény. Tzv. **KU-pásmo (KU-band)** pracuje s vyššími frekvencemi (12-14 GHz), což umožňuje používat rozměrově menší antény (o průměru cca 60-100 cm).

Pomocí družicových spojů lze vytvořit přenosové kanály a okruhy širokého spektra přenosových rychlostí (až desítek a stovek megabitů za sekundu), které se svými vlastnostmi plně vyrovnají pozemním kabelovým spojům, a v mnohém je i předčí. Družicové spoje však mají i své nevýhody. Tou nejvýraznější je zpoždění, které při přenosu vzniká.

Ze Země na geostacionární družici a zpět dorazí signál přibližně za 250 až 300 ms (podle místa na Zemi, odkud byl vyslán, a ve kterém byl přijat). Budeme-li čekat ještě na potvrzující odpověď, nemůže tato přijít dříve než na cca 500-600 milisekund. A to

je velmi dlouhá doba. Ne ani tak pro člověka, jako pro přenosové protokoly, které čekají na potvrzení příjmu od protější strany. Ty musí být vhodně uzpůsobeny, aby velké zpoždění potvrzující odpovědi neinterpretovaly chybně, a nesnažily se příliš brzy o opětovné vyslání již dříve přenesených bloků dat.

Družicové spoje lze využít pro realizaci dvoubodových spojení. Vzhledem ke své povaze však umožňují i přenosy od jednoho zdroje k více příjemcům (tzv. **broadcasting**, který využívají např. různé satelitní televize), a od více zdrojů k jedinému příjemci (tzv. **vícenásobný přístup, multiple access)**.

.PI OBR18_4A.TIF, 20, 40, 10 .PI OBR18_4B.TIF, 20, 40, 10

Obr. 18.4.: Datová síť VSAT;

a/ broadcasting

b/ vícenásobný přístup

Tyto dvě vlastnosti se s úspěchem využívají v datových sítích, vytvářených pomocí družicových spojů. Používají tak vysoké frekvence, že vystačí jen s relativně malými pozemními anténami resp. s velmi malým úhlem vyzařování. Jednotlivé uzlové body této sítě jsou označovány jako **terminály VSAT (Very Small Aperture Terminal)**.

Datová síť VSAT (VSAT Data Network) předpokládá existenci jednoho centrálního bodu (tzv. **hub**) a většího počtu terminálů VSAT, přičemž komunikace je možná jen mezi terminálem a centrálním bodem, nikoli přímo mezi dvěma terminály - viz obr. 18.4. Důvody jsou především technické a vyplývají ze samotné podstaty družicových spojů. Mnohým aplikacím, které sítě VSAT využívají, však takováto "centralizovaná" koncepce plně vyhovuje, neboť jde o aplikace, která umožňují velkému počtu uživatelů přístup k jednomu centralizovanému informačnímu zdroji - např. velké databázi, rezervačnímu systému apod.

Obecnou výhodou všech družicových spojů je jejich nezávislost na pozemní komunikační infrastruktuře, možná mobilita pozemních stanic, možnost realizovat spojení i do jakkoli odlehlých a nepřístupných míst, a v neposlední míře také vysoké přenosové rychlosti, kterých lze pomocí družicových spojů dosáhnout.

19/ Koaxiální kabel a kroucená dvoulinka

Jednou z odlišností lokálních a rozlehlých počítačových sítí je vztah jejich provozovatelů k prostředkům, které slouží k propojování uzlových počítačů. V případě rozlehlých sítí jde vesměs o přenosové cesty resp. okruhy či kanály, pouze pronajaté od spojové organizace, která je jejich vlastníkem.

V případě lokálních počítačových sítí si nezbytné prostředky propojení zřizují provozovatelé sítě sami, a tyto pak zůstávají jejich majetkem. Nejčastěji jde o různé druhy drátových přenosových cest (kabelů), v poslední době se však stále více prosazují také optické kabely.

Možnosti využití jednotlivých druhů kabelů jsou určeny především jejich kvantitativními parametry. K nejdůležitějším patří:

- přenosová rychlost, kterou lze na daném kabelu dosáhnout: u drátových přenosových cest se pohybuje v řádu jednotek až desítek Mbitů za sekundu, u optických kabelů je ještě vyšší.

- **útlum** (**attenuation**): představuje zeslabení přenášeného signálu. Je způsoben odporem, který kabel klade přenášenému signálu - bývá větší pro vyšší frekvence přenášeného signálu (a tedy pro vyšší přenosové rychlosti), a roste také se zmenšováním průměru kabelu. Celková hodnota útlumu je přímo úměrná délce kabelu a je jedním z rozhodujících faktorů, které určují maximální použitelnou délku souvislého úseku (segmentu) kabelu.

- odolnost vůči **rušení** (**interference**): v okolí kabelu může docházet k různým jevům, které mají nepříznivý vliv na přenášený signál - jde např. o provoz různých elektrických spotřebičů apod. Každý druh kabelu vykazuje obecně jinou odolnost vůči rušení.

Jednou ze specifických forem rušení je tzv. **přeslech** (**cross-talk**), kdy signál, přenášený jedním kabelem, ovlivňuje průběh signálu v jiném kabelu - nejčastěji se projevuje u souběžně vedených vodičů.

V důsledku útlumu a rušení pak dochází **zkreslení (distortion)** přenášeného signálu, které se u datových přenosů v konečné podobě projevuje příjmem jiných dat, než jaké byly skutečně vyslány.

Dnes zřejmě nejpopulárnější drátovou přenosovou cestou je tzv. **koaxiální kabel** (**coaxial cable**). Tvoří jej vnitřní vodič (měděný nebo postříbřený), kolem kterého je nanesena izolující vrstva dielektrika - viz obr. 19-1. Na této vrstvě je pak naneseno vodivé opletení, které je samo překryto další izolující vrstvou - vnějším pláštěm.

.PI OBR19_1.TIF, 20, 40, 10

Obr. 19-1.: a/ koaxialní kabel

b/ kroucená dvoulinka

Vodivé opletení představuje "rozprostřený" vodič, jehož podélná osa je shodná s osou vnitřního vodiče - proto označení "ko-axiální" (tj. souosý) kabel.

Hlavní efekt vodivého opletení spočívá především v odstínění vnitřního vodiče od vlivu vnějšího rušení - koaxiální kabely jsou proto vůči rušení dosti odolné. Vyrábí se v různých provedeních, které se liší rozměry, mechanickým provedením i hodnotou tzv. **impedance**, která vyjadřuje odpor, kladený střídavému signálu.

Koaxiální kabely se používají např. v lokálních sítích Ethernet. Zde je základním typem koaxiální kabel o vnějším průměru 10 mm, se čtyřnásobným opletením a impedancí 50 Ohmů. Souvislý segment tohoto kabelu může mít délku až 500 metrů, na větší vzdálenosti je pak nutné vzájemně spojovat více segmentů pomocí tzv. opakovačů (o nich si budeme povídat později). Formální označení tohoto kabelu je RG-11, v praxi je ale označován spíše jako **tlustý Ethernet** (**Thick Ethernet**), nebo také jako **žlutý kabel (Yellow Cable**) - vzhledem ke svému velkému průměru resp. své charakteristické barvě. Jeho výhodou je především značný dosah a velká odolnost proti rušení, nevýhodou pak vysoká cena a malá ohebnost.

.cp20

V mnoha konkrétních aplikacích však nejsou kvality tlustého Ethernetu dostatečně využity, a tak se zde používá tzv. **tenký Ethernet** (**Thin Ethernet**) - o stejné impedanci (50 Ohmů), polovičním průměru (4,9 mm), jen se dvojitým opletením a přibližně 5x lacinější, nebo tzv. **Cheapernet**, který má jen jednoduché opletení. Vyšší útlum těchto kabelů a jejich menší odolnost vůči rušení dovolují vytvářet souvislé segmenty jen do cca 185 metrů. Velkou roli hraje často také výrazně větší ohebnost "tenkého" Ethernetu, formálně označovaného jako RG-58. Všechny tři dosud uvedené druhy koaxiálních kabelů se v sítích Ethernet standardně používají pro přenosové rychlosti 10 Mbit/sekundu.

V sítích Arcnet se používá jiný druh koaxiálního kabelu, o impedanci 93 Ohmů (označovaný jako RG-62), který se používá také u sálových počítačů IBM pro připojování terminálů k terminálovým řadičům. Kromě toho se používá také koaxiální kabel o impedanci 75 Ohmů, převzatý z televizní techniky (označovaný jako CATV kabel).

Vedle koaxiálních kabelů připadají v úvahu pro propojování uzlových počítačů také dvojice běžných jednožilových vodičů. Aby se co možná nejvíce minimalizoval vliv rušení, používá se pro tyto vodiče tzv. diferenciální buzení - což znamená, že přenášenou informaci vyjadřuje rozdíl signálů na obou vodičích. Dojde-li pak vlivem vnějšího rušení ke stejné změně signálu na obou vodičích, na rozdílu obou signálu se to neprojeví.

Dva paralelně vedoucí vodiče však vždy fungují jako jednoduchá anténa. Při souběžném vedení více takovýchto dvojic pak může mezi nimi docházet k nežádoucímu přeslechu, tedy ke vzájemnému ovlivňování přenášených signálů. Lze jej výrazně omezit zkroucením jednotlivých dvojic vodičů, což minimalizuje jejich chování jako antény - tak vzniká tzv. **kroucená dvoulinka** (**Twisted Pair**) - viz obr. 19-1. b/.

Ještě větší odolnosti proti rušení lze dosáhnout stíněním dvojice vodičů - pak jde o tzv. **stíněnou kroucenou dvoulinku (Shielded Twisted Pair**), která je pak ovšem dražší než původní **nestíněná kroucená dvoulinka** (**Unshielded Twisted Pair**).

Kroucená dvoulinka se používá mj. také v telefonní technice - běžné telefonní přípojky jsou vesměs realizovány právě tímto způsobem. Pomocí kroucených dvoulinek jsou řešeny i telefonní rozvody v rámci budov, od pobočkové ústředny hvězdicovitě do jednotlivých bytů, kanceláří apod. V rámci existujících telefonních instalací, zvláště těch, které byly prováděny v nedávné době, mohlo zůstat mnoho nevyužitých párů, původně zamýšlených jako rezerva. To je výzva, kterou tvůrci lokálních sítí nemohli nechat bez odezvy.

V současné době již lze stíněnou dvoulinku používat v různých typech lokálních sítí. Pro sítě Ethernet byl v nedávné době přijat standard, definující způsob použití kroucené dvoulinky telefonního typu pro přenosy rychlostí 10 Mbit za sekundu. Tedy stejně rychle, jako po koaxiálním kabelu. Rozpracován je také standard pro FDDI přenos pro kroucené dvoulince rychlostí 100 Mbit za sekundu (označovaný také jako standard CDDI nebo Copper Distributed Data Interface).

20/ Optické kabely

Jak jsme si již uvedli v 5. dílu našeho seriálu, je dosažitelná přenosová rychlost závislá především na šířce pásma přenosového kanálu, tedy na rozsahu frekvencí, které je tento kanál schopen přenést.

Chceme-li dosahovat velmi velkých přenosových rychlostí, musíme samozřejmě volit takové způsoby přenosu, které mají šířku přenášeného pásma co možná největší. Je přitom vcelku zřejmé, že velké šířky přenášeného pásma lze dosáhnout nejsnáze tam, kde jsou frekvence přenášených signálů velmi vysoké.

Z tohoto pohledu je velmi lákavé používat pro přenos dat např. viditelné světlo, které má frekvenci přibližně 10^8 MHz, a vzhledem k tomu skýtá opravdu nebývalé možnosti.

Přenášená číslicová data můžeme reprezentovat pomocí světelných impulsů - přítomnost impulsu může představovat např. logickou 1, zatímco jeho nepřítomnost logickou 0. Pro praktickou realizaci potřebujeme ovšem celý optický přenosový systém, složený ze zdroje, přenosového média a přijímače - viz obr. 20.1 a/.

.PI OBR20_1.TIF, 20, 40, 10

Obr. 20.1.: a/ Optický přenosový systém

b/ Numerická apertura

Vlastním zdrojem světla může být obyčejná elektroluminiscenční dioda (**dioda LED**, **Light Emitting Diode)** nebo nákladnější **laserová dioda** (**laser diode**), které emitují světelné pulsy na základě přiváděného proudu. Detektorem na straně přijímače pak bývá **fotodioda** (**photodiode**), která naopak převádí dopadající světelné impulsy na elektrické signály.

Úkolem přenosového média je dopravit světelný paprsek od jeho zdroje k detektoru - s co možná nejmenšími ztrátami. K tomuto účelu se používá tenké **optické vlákno** (**optical fiber**), tvořené **jádrem** (core) a **pláštěm** (cladding). Jádro má nejčastěji průměr v řádu jednotek až desítek mikrometrů, a je vyrobené nejčastěji z různých druhů skla, eventuelně i z plastu.

Pro pochopení způsobu, jakým je světelný paprsek optickým vláknem veden, je nutné si nejprve uvědomit jeden základní poznatek z oblasti fyziky:

dopadá-li světelný paprsek na rozhraní dvou prostředí s různými optickými vlastnostmi (např. na rozhraní mezi jádrem a pláštěm), v obecném případě se část tohoto paprsku odráží zpět do původního prostředí, a část prostupuje do druhého prostředí. Záleží však na úhlu, pod jakým paprsek dopadá na rozhraní (měřeném od kolmice na místo dopadu). Je-li tento úhel větší než určitý mezní úhel (měřený od kolmice na místo dopadu a daný optickými vlastnostmi obou prostředí), dochází k úplnému odrazu paprsku zpět do původního prostředí - viz obr. 20.1 b/.

V důsledku opakovaných úplných odrazů, které probíhají bez jakýchkoli ztrát, pak světelný paprsek sleduje dráhu jádra optického vlákna - je tímto jádrem veden.

Rozmezí úhlů, pod kterými může světelný paprsek dopadat na optické vlákno tak, aby byl veden, definuje tzv. **numerickou aperturu -** viz obr. 20.1 b/.

Způsob, jakým optické vlákno paprsek vede, záleží také na tom, jak se mění optické vlastnosti (konkrétně tzv. **index lomu - refraction index**) na přechodu mezi jádrem vlákna a jeho pláštěm. Mění-li se skokem a je-li průměr jádra dostatečně velký (50-

100 mikrometrů), jde o vlákno, schopné vést různé vlny světelných paprsků - tzv. **vidy** (**modes**). Jde tedy o **mnohovidové vlákno** (**multimode fiber**), v tomto případě se stupňovitým indexem lomu (step index fiber) - viz obr.20.2 a/.

Pokud se index lomu na přechodu mezi jádrem vlákna a jeho pláštěm nemění skokem, ale plynule, jde o mnohovidové vlákno s tzv. gradientním indexem lomu (graded index fiber), které přenášené vidy ohýbá - viz obr. 20.2 b/.

Výhodou mnohovidových vláken je relativně nízká cena, snažší spojování, velká numerická apertura a možnost buzení luminiscenční diodou.

.PI OBR20_2.TIF, 20, 40, 10

Obr. 20.2.:

a/ Mnohovidové vlákno se stupňovitým indexem lomu

b/ Mnohovidové vlákno s gradientním indexem lomu

c/ Jednovidové vlákno

Nejvyšších přenosových rychlostí (až Gigabity/sekundu na vzdálenosti do 1 km) lze dosáhnout na tzv. **jednovidových vláknech** (**single mode fiber**), které přenáší jen jediný vid - viz obr. 20.2 c/. Schopnosti vést jediný vid bez odrazů i ohybů se dosahuje buďto velmi malým průměrem jádra (řádově jednotky mikrometrů), nebo velmi malým poměrným rozdílem indexů lomu jádra a jeho pláště. V každém případě jsou jednovidová vlákna dražší než mnohovidová, lze je ovšem použít pro přenosy na delší vzdálenosti (až 100 km bez opakovače), než vlákna mnohovidová. Pro své buzení však již vyžadují laserové diody.

Optická vlákna jsou velmi citlivá na mechanické namáhání a ohyby. Jejich ochranu proto musí zabezpečovat svým konstrukčním řešením **optický kabel**, který kromě jednoho či více optických vláken obvykle obsahuje i vhodnou výplň, zajišťující potřebnou mechanickou odolnost.

Kromě velké přenosové rychlosti je další velkou výhodou optických vláken jejich naprostá necitlivost vůči elektromagnetickému rušení (což je velmi důležité např. v průmyslových aplikacích). Výhodou je také velká bezpečnost proti odposlechu, malý průměr a malá hmotnost optických kabelů. Poněkud problematičtější je spojování jednotlivých vláken, technologie jejich lepení či svařování však již jsou v praxi dostatečně zvládnuty.

Pro počítačové sítě jsou optická vlákna atraktivní především pro vysokou přenosovou rychlost, kterou umožňují dosáhnout s poměrně nízkými náklady. Jde tedy o technologii velmi perspektivní (a to nejen pro počítačové sítě). V současné době již existují dva standardy, které se týkají použití optických vláken v počítačových sítích: FDDI (Fiber Distributed Data Interface) pro lokální sítě typu token ring, s přenosovou rychlostí 100 Mbit/sekundu, a DQDB (Distributed Queue Dual Bus) pro tzv. metropolitní sítě, s přenosovou rychlostí až 155 Mbit/sekundu.

21/ Standard, norma, doporučení

Výrobci výpočetní techniky se dlouhou dobu snažili prosadit na trhu se strategií, kterou lze charakterizovat jako snahu "být pro všechny vším". Snažili se dosáhnout

toho, aby si zákazník kupoval všechno od nich - jak hardware se všemi potřebnými doplňky, tak i nezbytný software, školení uživatelů, servis atd.

V době sálové výpočetní techniky byla takováto strategie ještě realizovatelná, neboť zájmy odběratelů nebyly tak diversifikované a specificky zaměřené jako dnes, a také požadavky na vzájemnou spolupráci jednotlivých počítačů nebyly příliš velké.

Výrobci, kteří se příliš neohlíželi na sebe navzájem, si mohli dovolit koncipovat své produkty výhradně podle svých vlastních představ a uplatnit v nich taková řešení, která sami považovali za nejvhodnější - ke kterým dospěli na základě vlastních zkušeností, tradicí, vlastního výzkumu a vývoje - a která je činila neslučitelnými a neschopnými spolupráce s produkty jiných výrobců.

Tím ovšem vznikla specifická forma vázanosti na jediného výrobce, daná nikoli jeho monopolním postavením, ale neslučitelností jím používaného technického řešení s tím, které používají jiní výrobci. V angličtině se takovéto řešení označuje přívlastkem **proprietary**, stejně tak jako produkty, které z tohoto řešení vycházejí.

Prosadit vlastní řešení, a to ještě se ziskem, si však v dlouhodobém výhledu mohly dovolit jen ty největší firmy. Menší firmy, které nemohly nést stále větší náklady na vývoj a marketing vlastních řešení, se ve vlastním zájmu musely přizpůsobit těm řešením, které si zvolily velké firmy. Nešlo přitom ani tak o převzetí technologií či výrobních postupů (které jsou často pečlivě chráněné), jako spíše o převzetí konvencí, parametrů a protokolů, s cílem zajistit **kompatibilitu** (slučitelnost) vlastních produktů s produkty jiných výrobců. Názorným příkladem může být architektura osobních počítačů PC - zde se prakticky všichni výrobci přizpůsobili řešení, které si podle svého zvolila jediná mamutí firma - IBM.

Řešení, kterému se přizpůsobují různí výrobci a které tak představuje určitou společnou konvenci, zajišťující vzájemnou kompatibilitu produktů od různých výrobců, si již zaslouží přívlastek **standardní**, jako protipól anglického **proprietary**. Samotný obsah resp. podstata tohoto řešení se pak v širším slova smyslu označuje jako **standard.**

Standardní řešení resp. standard může vniknout tak, že se z podnětu či pod záštitou určité instituce, která je k tomu příslušná, sejde skupina odborníků a vypracuje návrh příslušného řešení. Ten je posléze kodifikován (tj. dostane formu oficiálního dokumentu příslušné instituce), a pak je prosazován do praxe. Podstatné přitom je, že zmíněné standardizační instituce obvykle nereprezentují přímo jednotlivé výrobce (i když tito se na jejich práci mohou významně podílet).

Standard, který je kodifikován, je standardem **de jure**. Jeho závaznost pro výrobce i uživatele je ovšem různá podle toho, jaký má právní statut resp. jaký je statut toho, kdo jej formálně vydává.

Například standardy, vypracované a vydávané mezinárodními standardizačními institucemi, mají často pouze formu **doporučení** a po formální stránce nejsou právně závazné. Právní závaznost pak mívají až návrhy ve formě **norem**, které v rámci jednotlivých zemí vypracovávají k tomu oprávněné instituce, často na základě doporučení, přijatých mezinárodními organizacemi.

Samotní výrobci nemají možnost vydávat standardy *de jure,* neboť obvykle nemohou vydávat oficiální doporučení či dokonce normy, závazné pro jiné výrobce. Pokud

jejich vlastní řešení spontánně a na dobrovolném základě přebírají i jiní výrobci, stává se toto řešení standardardem **de facto**. Příkladem takovéhoto standardu může být jazyk pro ovládání modemů (tzv. řídící jazyk AT, viz desátý díl našeho seriálu), který u svých produktů zavedla firma Hayes, a ostatní výrobci se mu přizpůsobili.

Jsou ovšem i případy, kdy se vlastní ("proprietary") řešení určitého výrobce může stát "oficiálním" standardem (standardem *de jure*). Jde o taková řešení, která se ukáží být natolik životaschopná, že se nejprve stanou standardy *de facto*, a posléze je příslušné standardizační instituce převezmou jako "své" standardy - buď bez jakýchkoli změn, nebo s určitými úpravami. Příkladem může být koncepce sítí Ethernet, která původně vznikla jako vlastní řešení firmy Xerox, záhy se stala standardem *de facto*, a posléze se s drobnými úpravami stala i standardem *de jure* (standardem IEEE 802.3).

Jestliže v oblasti počítačů mají standardy neobyčejný význam, v oblasti komunikací a počítačových sítí jsou přímo životně důležité. Umožňují vzájemnou spolupráci nejrůznějších komunikačních i výpočetních prostředků od různých výrobců, stejně tak jako jejich vazbu na přenosové prostředky spojových organizací a jejich využití. Týkají se prakticky všech aspektů komunikací a činností sítí - od typu používaných konektorů a jejich zapojení, přes způsob modulace přenášených signálů až např. po formát zpráv elektronické pošty.

22/ Kdo je kdo - ve světě standardů

Mezinárodní organizace, které se zabývají tvorbou standardů (standardů *de jure*, viz minule), obvykle spadají do dvou kategorií - jsou to jednak organizace, vytvořené na základě mezivládních dohod zúčastněných států, a jednak dobrovolné organizace, vytvořené na jiném základě. V oblasti počítačových sítí jde především o dvě organizace - CCITT a ISO, po jedné z každé kategorie.

Pro koordinaci telekomunikací v celosvětovém měřítku byla v rámci OSN vytvořena **Mezinárodní telekomunikační unie** (**ITU - International Telecommunications Union**) jako odborná mezinárodní organizace. Má tři složky - dvě se zabývají oblastí rozhlasového a televizního vysílání, zatímco třetí má na starosti otázky telefonie a komunikačních systémů pro přenos dat. Touto třetí složkou je **Mezinárodní poradní sbor pro telegraf a telefon**, známější spíše pod zkratkou **CCITT** (**Commité Consultatif International de Télegraphique et Téléphonique**).

Členy CCITT jsou především spojové organizace, které v jednotlivých členských zemích odpovídají za oblast telekomunikací (a mají na to vesměs ze zákona monopol). V některých zemích to jsou podniky řízené státem, v jiných zemích to jsou přímo vládní instituce - označované souhrnnou zkratkou **PTT** (**Post, Telegraph & Telephone**). Kromě toho mohou být členy CCITT i další organizace, ovšem již bez hlasovacího práva.

Standardy, které organizace CCITT vydává, mají formu doporučení (nikoli norem), a jsou schvalovány na plenárních zasedáních, která se konají jednou za čtyři roky - z tohoto důvodu jsou jednotlivá doporučení vydávána ve čtyřletých cyklech (jubilejní desáté plenární zasedání se bude konat právě v letošním roce).

Přípravou návrhů jednotlivých doporučení se v rámci CCITT zabývají tematicky zaměřené **studijní skupiny** (**SG, Study Group**). Vlastní doporučení CCITT se

označují písmenem a číslem, přičemž písmeno odpovídá studijní skupině, která návrh doporučení připravila, a tím současně i tematickému zaměření doporučení. Přehled studijních skupin, relevantních pro oblast počítačových sítí, uvádí tabulka 21.1.

Mezi nejznámější standardy v oblasti počítačových sítí, které pochází od organizace CCITT, patří např. doporučení X.25 pro veřejné datové sítě, doporučení V.21, V.22, V.22 bis a další pro modemy atd.

.cp20

Skupina	Tematické zaměření	Doporučení
SG VII	Datové sítě	X.nnn
SG VIII	Koncová zařízení telematických služeb	S.nnn, T.nnn
SG IX	Telegrafní sítě a koncová zařízení	R.nnn, W.nnn
SG XVII	Přenos dat po telefonní síti	V.nnn
SG XVIII	Číslicové sítě	G.nnn

Tab. 22.1.: Některé studijní skupiny CCITT

Reprezentantem druhé skupiny standardizačních organizací nevládního charakteru je **Mezinárodní organizace pro standardizaci** (**ISO - International Standards Organization**, formálně správný název však zní: **International Organization for Standardization**).

ISO je dobrovolnou nevládní organizací. Jejími členy jsou národní organizace, zabývající se standardizací v jednotlivých zemích. Své standardy vydává organizace ISO sice formálně jako normy, ty však stejně jako doporučení CCITT nemusí být pro jednotlivé členské země závazné. Jejich nedodržování je ale spíše výjimkou.

V rámci organizace ISO existuje přibližně 200 **technických komisí** (**TC, Technical Committee**), které se dále člení na **podkomise** (**SC, Subcommittees**), které se skládají z **pracovních skupin (WG, Workgroups**).

Vlastní návrhy standardů jsou připravovány v pracovních skupinách (WG), které nejprve vytvoří **koncept návrhu** (**DP, Draft Proposal**). Ten je rozdán všem zúčastněným organizacím, které pak mají šest měsíců na jeho připomínkování. Pokud koncept návrhu získá podporu svých posuzovatelů, je po zapracování připomínek vytvořen **koncept normy** (**DIS, Draft International Standard**), který je opět rozdistribuován a stává se základem pro hlasování o svém přijetí. V případě úspěchu pak vzniká **norma** (**International Standard**).

V oblasti počítačových sítí je snad nejznámějším standardem organizace ISO její tzv. referenční model OSI (o kterém si budeme povídat podrobněji).

Oblast, kterou pokrývají standardy resp. normy ISO, je velmi široká, a zahrnuje i "nepočítačovou" tematiku. V oblasti telekomunikací se překrývá se "sférou zájmu"

CCITT - obě organizace zde naštěstí dokáží spolupracovat alespoň natolik, aby se vyhnuly existenci vzájemně neslučitelných standardů. Organizace ISO je dokonce členem CCITT (ovšem bez hlasovacího práva).

.cp9

Velmi významné jsou ovšem i některé národní organizace, které se zabývají standardizací v jednotlivých zemích. V USA jsou takovéto organizace hned tři. První je **NIST** (**National Institute of Standards and Technology**, dříve **NBS** resp. **National Bureau of Standards**), která je účelovým orgánem ministerstva obchodu USA, a vydává normy pro oblast působnosti federální vlády. V Evropě známější je nevládní nevýdělečná organizace **ANSI** (**American National Standards Institute**), jejímiž členy jsou výrobci, poskytovatelé spojových služeb a další zainteresované organizace z USA. Organizace ANSI je reprezentantem USA v ISO, a mnohé ANSI standardy jsou organizací ISO přejímány a stávají se mezinárodními standardy (tj. normami ISO). Nejznámějším standardem, který pochází od organizace ANSI, je bezesporu znakový kód ASCII.

Třetí národní standardizační organizací v USA je **EIA** (**Electronic Industries Assocation)**, sdružující ve svých řadách především výrobce zařízení pro telekomunikace. Snad nejznámějším standardem, který pochází od této organizace, je standard RS-232-C pro seriové rozhraní mezi terminálem a modemem, převzatý s minimálními úpravami i jako standard CCITT pod označením V.24

Standardy však mohou pocházet i od různých profesních sdružení. Příkladem zde může být prestižní americká společnost elektrotechnických a elektronických inženýrů **IEEE** (**Institute of Electrical and Electronics Engineers**), která kromě vydávání odborných časopisů a pořádání konferencí má i vlastní orgán, zabývající se tvorbou standardů. Pro oblast lokálních počítačových sítí je z nich nejvýznamnější především skupina standardů IEEE 802, převzatá i organizací ISO jako norma ISO 8802.

23/ Vrstvy, vrstvové modely, protokoly

Řešit určitý problém jako celek je obvykle možné jen po určitou hranici - dokud rozsah problému resp. jeho složitost nepřesáhne schopnosti řešitele či řešitelů a možnosti nástrojů a metod, které přitom používají.

Jakmile se určitý problém stává příliš složitým, je vhodné provést jeho dekompozici - rozdělit jej na několik dílčích problémů, které by bylo možné řešit samostatně a nezávisle na sobě.

Dobrým příkladem problému, který je příliš velký na to, aby bylo únosné jej řešit jako jediný velký problém, je otázka zajištění základních funkcí počítačové sítě. Jistě si lze snadno představit, že se jedná především o otázku základního programového vybavení sítě, které bezprostředně ovládá technické prostředky sítě - síťový hardware, tj. nejrůznější komunikační vybavení, uzlové počítače atd. - a jejich prostřednictvím zajišťuje chod celé sítě jako takové.

.PI OBR23_1.TIF, 20, 40, 10

Obr. 23.1: Představa vzájemného vztahu mezi vrstvami

Vzhledem k charakteru počítačových sítí a k povaze úkolů, které je třeba zajistit, se jako nejvhodnější ukázala být dekompozice základního programového vybavení sítě na hierarchicky uspořádané **vrstvy** (**layers**). Každá vrstva má na starosti zajištění

přesně vymezeného okruhu úkolů. Mechanismy, pomocí kterých tyto úkoly zajišťuje, pak nabízí k využití jako své služby vrstvě bezprostředně vyšší. Například vrstva, která zajišťuje přenos jednotlivých bitů, může nabízet své služby bezprostředně vyšší vrstvě, která s jejich pomocí přenáší celé bloky dat apod. V obecném případě (viz obr. 23.1.) tedy každá vrstva nabízí určitý repertoár služeb vrstvě bezprostředně vyšší, a k realizaci těchto služeb sama využívá služeb vrstvy bezprostředně nižší.

Představa o tom, kolik samostatných vrstev vymezit a jaké jim svěřit úkoly, tvoří tzv. **vrstvový model** (**layered model**). Názorů na to, jak by takový vrstvový model měl vypadat, může být samozřejmě celá řada. Přístě se seznámíme s jedním z nich, kterým je tzv. referenční model ISO/OSI.

Rozdělení na hierarchické vrstvy v rámci vrstvového modelu ještě samo o sobě nezajišťuje hlavní efekt dekompozice jednoho velkého problému na několik dílčích problémů - možnost řešit tyto dílčí problémy samostatně a nezávisle na sobě. K tomu je nutné ještě stanovit přesná pravidla vzájemné součinnosti sousedních vrstev - tedy definovat přesná **rozhraní** (**interface**) mezi jednotlivými vrstvami. Součástí této definice musí být např. přesné vymezení jednotlivých služeb, způsob jejich volání, počty parametrů atd.

Jakmile jsou známy úkoly, které má určitá vrstva řešit, a je také přesně definováno její rozhraní s oběma sousedními vrstvami, je možné začít uvažovat o způsobu, jak zajistit ty úkoly, které byly vrstvě svěřeny. Zde je dobré si uvědomit, že každá vrstva sice využívá služby vrstvy bezprostředně nižší a sama nabízí své služby vrstvě bezprostředně vyšší, jejím partnerem při komunikaci v síti je ale ta vrstva, která se na jiném uzlovém počítači nachází na stejné úrovni hierarchie vrstev (v angličtině se pro takovou vrstvu používá označení "**peer**"). Tyto stejnolehlé vrstvy musí být spolu domluveny na společných pravidlech vzájemné komunikace, které také musí důsledně dodržovat. Použijeme-li již dříve citovaný příklad vrstvy, která zajišťuje přenos celých bloků dat, jejím partnerem je stejnolehlá vrstva jiného uzlového počítače, se kterou musí být dohodnuta mj. na formátu bloků, které si navzájem posílají, na jejich délce, způsobu zabezpečení, způsobu reakce na příjem poškozeného bloku apod.

Soubor pravidel, které stejnolehlé vrstvy vrstvového modelu používají pro vzájemnou komunikaci, tvoří tzv. **protokol.**

Ke každé vrstvě v rámci určitého vrstvového modelu se tedy vztahuje určitý protokol, podle kterého příslušná vrstva pracuje. Pro jednu a tutéž vrstvu však může připadat v úvahu více různých protokolů - stejné úkoly totiž mohou být často zajišťovány různými způsoby.

.cp20

K určitému vrstvovému modelu, který definuje způsob rozčlenění na vrstvy, tedy může existovat celá **soustava protokolů** (**protocol suite**), v rámci které může pro některé vrstvy připadat v úvahu několik vzájemně alternativních protokolů. Například pro naše dvě citované vrstvy (zajišťující přenos bitů resp. celých bloků) bude obvykle existovat více různých protokolů podle toho, zda jsou k přenosu využívány např. komutované linky veřejné telefonní sítě, pevné telefonní okruhy, veřejná datová síť, optická vlákna, družicové spoje atd.

Jakmile je pro každou vrstvu vybrán jeden konkrétní protokol, vzniká tzv.**sestava protokolů** (**protocol stack**), která přesně odpovídá hierarchickému členění na jednotlivé vrstvy.

Navrhnout rozčlenění základního programového vybavení na jednotlivé vrstvy, vymezit úkoly, které by tyto vrstvy měly vykonávat a stanovit protokoly, které by se přitom měly používat - to vše již dává dosti ucelenou představu o tom, jak by počítačová síť měla vypadat a jak by měla fungovat. Na základě této ucelené představy, která tvoří **architekturu sítě** (**network architecture**), je již možné uvažovat o konkrétní implementaci.

Rozdělení základního programového vybavení sítě na vrstvy a jejich samostatná a nezávislá realizace má vedle rozdělení jednoho velkého problému na několi menších, snáze zvládnutelných částí, ještě jednu velmi významnou výhodu. Tou je možnost snažšího přizpůsobení změnám - stačí vždy vyměnit jen ty vrstvy, kterých se změna týká, a ostatní ponechat nezměněné.

24/ Referenční model ISO/OSI - jeho vznik

V sedmdesátých letech se začínají objevovat první významnější rozlehlé počítačové sítě. Vedle experimentálních sítí (jako např. ARPANET a CYCLADES), to byly především sítě, budované podle vlastních koncepcí předních výrobců počítačů (např. sítě SNA firmy IBM, DNA firmy DEC). Zatímco experimentální sítě byly od začátku heterogenní a přímo tedy počítaly se zapojováním počítačů různých typů od různých výrobců, možnost zapojení do ostatních sítí byla vázána na vlastnictví produktů příslušného výrobce, který si v rámci vlastní síťové architektury vytvářel své specifické konvence a protokoly.

Není proto divu, že záhy vyvstala naléhavá potřeba jednotného standardu pro vzájemné propojování počítačových systémů různých typů a koncepcí, pocházejících od různých výrobců. Potřebu vytvoření takovéhoto standardu si brzy uvědomila i mezinárodní organizace ISO, která se v roce 1977 rozhodla převzít tento úkol na svá bedra.

Vytvořila si k tomu novou podkomisi (SC16) v rámci své technické komise pro zpracování dat (TC97**,** viz 22. díl našeho seriálu), a tu pověřila přípravou nového standardu. Ten také již dostal první pracovní název:**Open Systems Architecture** (tj. architektura otevřených systémů). Přívlastek **"otevřený" (open)** zde měl zdůraznit, že systém, vyhovující zamýšlenému standardu, bude připraven pro vzájemné propojení se všemi ostatními systémy na celém světě, které budou vyhovovat témuž standardu.

První pracovní schůzka podskupiny SC16 se konala v březnu 1978. Všichni zúčastnění se rychle shodli na vrstvové koncepci, která nejen dokáže vyhovět všem stávajícím nárokům na vzájemné propojování otevřených systémů, ale je schopná i pozdějšího rozšiřování tak, aby mohla vyhovět i budoucím požadavkům. Podskupina SC 16 se proto začala okamžitě zabývat otázkou, kolik vrstev bude nejvhodnější uvažovat, a jaké úkoly těmto vrstvám svěřit.

Pro potřeby svého rozhodování si podskupina SC16 zformulovala celkem 13 principů, které se pak snažila aplikovat. Lze je shrnout do následujících myšlenek:

- samostatná vrstva by měla vzniknout všude tam, kde je zapotřebí jiný stupeň abstrakce

- každá vrstva by měla zajišťovat přesně vymezené funkce .Tyto funkce by měly být voleny tak, aby pro jejich realizaci mohly být vytvořeny standardizované protokoly s mezinárodní působností.

- rozhraní mezi vrstvami by měla být volena tak, aby byl minimalizován tok dat přes tato rozhraní

- počet vrstev by měl být tak velký, aby vzájemně odlišné funkce nemusely být zařazovány do stejné vrstvy, a současně s tím tak malý, aby celá architektura zůstala dostatečně přehledná.

Výsledkem aplikace těchto principů bylo vymezení sedmi vrstev a specifikace úkolů, které by tyto vrstvy měly zajišťovat. (Podrobněji se jim budeme zabývat v dalších pokračováních)

Ještě v průběhu práce na připravovaném standardu změnila podskupina SC16 i jeho pracovní název, z **Open Systems Architecture** na **Open Systems Interconnection Architecture**, doslova tedy: architektura vzájemného propojování otevřených systémů. Důvodem byla zřejmě skutečnost, že se celá práce skupiny zaměřila pouze na otázky propojování a "vnějšího chování" vzájemně propojených systémů, a nijak neřešila to, jak mají tyto systémy fungovat "uvnitř".

Vlastní návrh standardu byl dokončen za necelých 18 měsíců. V červenci roku 1979 byl předán jako koncept návrhu (draft proposal - viz 22. díl našeho seriálu) nadřízené technické komisi TC97, která jej koncem roku 1979 přijala za základ pro další vývoj navazujících standardů pro vzájemné propojování otevřených systémů v rámci organizace ISO.

Na poslední chvíli však byl ještě jednou změněn slovní název standardu - z blíže neznámých důvodů bylo vypuštěno slovo Architecture, a naopak byla přidána dvě nová slova: **Reference Model** neboli **referenční model.** Ta mají zdůraznit, že celý standard není jedním konkrétním návrhem způsobu, jak řešit vzájemné propojování, ale spíše společným rámcem či vzorem (referenčním modelem), podle kterého by mělo být vzájemné propojování systémů řešeno.

Správný název celého standardu je tedy: **Reference Model of Open Systems Interconnection** (Referenční model propojování otevřených systémů), a jako norma ISO má číslo 7498. V praxi se obvykle označuje zkratkou **RM OSI** nebo jen **ISO/OSI**, což současně zdůrazňuje jeho vztah k organizaci ISO (byl však současně převzat i organizací CCITT jako její standard X.200).

Zdůrazněme si znovu, že součástí referenčního modelu ISO/OSI je skutečně pouze vymezení jednotlivých vrstev a spefifikace úkolů, které by tyto vrstvy měly řešit. Referenční model nespecifikuje žádné konkrétní protokoly, pomocí kterých by funkce jednotlivých vrstev měly být realizovány. Nedefinuje dokonce ani přesné rozhraní mezi jednotlivými vrstvami - tedy konkrétní služby, které vrstvy bezprostředně nižší poskytují bezprostředně vyšším vrstvám.

.cp9

Protokoly a služby pro jednotlivé vrstvy vznikaly až dodatečně, a to jako samostatné standardy (resp. normy) ISO, nebo jako převzaté standardy jiných organizací (např. CCITT či IEEE). Jelikož organizace ISO striktně rozlišuje mezi protokolem a službou (na rozdíl od ostatních standardizačních organizací, jako např. CCITT, které takovéto

rozlišování nepoužívají), vznikaly původní standardy ISO vždy ve dvojicích - jeden standard pro protokol, a druhý pro služby, poskytované prostřednictvím tohoto protokolu. Postupně tak vznikaly resp. stále vznikají či jsou přejímány protokoly, definující možné způsoby fungování jednotlivých vrstev ISO/OSI modelu.

25/ Referenční model ISO/OSI - sedm vrstev

Jak jsme si uvedli v předcházejícím dílu našeho seriálu, dospěli tvůrci referenčního modelu ISO/OSI při svých úvahách k závěru, že optimální počet vrstev síťového softwaru bude sedm. Jaké vrstvy to jsou a jaké úkoly majířešit? Vezměme to postupně zdola nahoru, od nejnižší vrstvy k nejvyšší:

1. Fyzická vrstva (Physical Layer)

Úkol této vrstvy je zdánlivě velmi jednoduchý - zajistit přenos jednotlivých bitů mezi příjemcem a odesilatelem prostřednictvím fyzické přenosové cesty, kterou tato vrstva bezprostředně ovládá. K tomu je ovšem třeba vyřešit mnoho otázek převážně technického charakteru - např. jakou úrovní napětí bude reprezentována logická jednička a jakou logická nula, jak dlouho "trvá" jeden bit, kolik kontaktů a jaký tvar mají mít konektory kabelů, jaké signály jsou těmito kabely přenášeny, jaký je jejich význam, časový průběh apod. Problematika fyzické vrstvy proto spadá spíše do působnosti elektroinženýrů a techniků.

2. Linková vrstva (Data Link Layer)

Fyzická vrstva poskytuje jako své služby prostředky pro přenos jednotlivých bitů. Bezprostředně vyšší linková vrstva (někdy nazývaná též: **spojová vrstva** či **vrstva datového spoje**) pak má za úkol zajistit pomocí těchto služeb bezchybný přenos celých bloků dat (velikosti řádově stovek bytů), označovaných jako **rámce** (**frames**). Jelikož fyzická vrstva nijak neinterpretuje jednotlivé přenášené bity, je na linkové vrstvě, aby správně rozpoznala začátek a konec rámce, i jeho jednotlivé části.

Na přenosové cestě může docházet k nejrůznějším poruchám a rušení, v jejichž důsledku jsou přijaty jiné hodnoty bitů, než jaké byly původně vyslány. Jelikož fyzická vrstva se nezabývá významem jednotlivých bitů, rozpozná tento druh chyb až linková vrstva. Ta kontroluje celé rámce, zda byly přeneseny správně (podle různých kontrolních součtů, viz 3. díl našeho seriálu). Odesilateli potvrzuje přijetí bezchybně přenesených rámců, zatímco v případě poškozených rámců si vyžádá jejich opětovné vyslání.

3. Síťová vrstva (Network Layer)

Linková vrstva zajišťuje přenos celých rámců, ovšem pouze mezi dvěma uzly, mezi kterými vede přímé spojení. Co ale dělat, když spojení mezi příjemcem a odesilatelem není přímé, ale vede přes jeden či více mezilehlých uzlů? Pak musí nastoupit síťová vrstva, která zajistí potřebné **směrování** (**routing**) přenášených rámců, označovaných nyní již jako **pakety** (**packets**). Síťová vrstva tedy zajišťuje volbu vhodné trasy resp. cesty (**route**) přes mezilehlé uzly, a také postupné předávání jednotlivých paketů po této trase od původního odesilatele až ke konečnému příjemci.

Síťová vrstva si tedy musí "uvědomovat" konkrétní topologii sítě (tj. způsob vzájemného přímého propojení jednotlivých uzlů).

4. Transportní vrstva (Transport Layer)

Síťová vrstva poskytuje bezprostředně vyšší vrstvě služby, zajišťující přenos paketů mezi libovolnými dvěma uzly sítě. Transportní vrstvu proto zcela odstiňuje od skutečné topologie sítě a vytváří jí tak iluzi, že každý uzel sítě má přímé spojení s kterýmkoli jiným uzlem sítě.

Transportní vrstvě díky tomu stačí zabývat se již jen komunikací koncových účastníků (tzv. **end-to-end** komunikací) - tedy komunikací mezi původním odesilatelem a konečným příjemcem.

Při odesílání dat zajišťuje transportní vrstva sestavování jednotlivých paketů, do kterých rozděluje přenášená data, a při příjmu je zase z paketů vyjímá a skládá do původního tvaru. Dokáže tak zajistit přenos libovolně velkých zpráv, přestože jednotlivé pakety mají omezenou velikost.

5. Relační vrstva (Session Layer)

Úkolem této vrstvy je navazování, udržování a rušení **relací** (**sessions**) mezi koncovými účastníky. V rámci navazování relace si tato vrstva vyžádá na transportní vrstvě vytvoření spojení, prostřednictvím kterého pak probíhá komunikace mezi oběma účastníky relace. Pokud je třeba tuto komunikaci nějak řídit (např. určovat, kdo má kdy vysílat, nemohou-li to dělat oba účastníci současně), zajišťuje to právě tato vrstva, která má také na starosti vše, co je potřeba k ukončení relace a zrušení existujícího spojení.

6. Prezentační vrstva (Presentation Layer)

Data, která se prostřednictvím sítě přenáší, mohou mít mj. povahu textů, čísel či obecnějších datových struktur. Jednotlivé uzlové počítače však mohou používat odlišnou vnitřní reprezentaci těchto dat - např. střediskové počítače firmy IBM používají znakový kód EBCDIC, zatímco většina ostatních pracuje s kódem ASCII. Podobně jeden počítač může zobrazovat celá čísla v doplňkovém kódu, zatímco jiný počítač v přímém kódu apod. - potřebné konverze přenášených dat má na starosti právě tato prezentační vrstva.

V rámci této vrstvy bývá také realizována případná komprese přenášených dat, eventuálně i jejich šifrování.

7. Aplikační vrstva (Application Layer)

Koncoví uživatelé využívají počítačové sítě prostřednictvím nejrůznějších síťových aplikací - systémů elektronické pošty, přenosu souborů, vzdáleného přihlašování (remote login) apod. Začleňovat všechny tyto různorodé aplikace přímo do aplikační vrstvy by pro jejich velkou různorodost nebylo rozumné. Proto se do aplikační vrstvy zahrnují jen části těchto aplikací, které realizují společné resp. obecně použitelné mechanismy. Uvažujme jako příklad právě elektronickou poštu - ta její část, která zajišťuje vlastní předávání zpráv v síti, je součástí aplikační vrstvy. Na všech uzlových počítačích, které používají tentýž systém elektronické pošty, je tato část stejná. Uživatelské rozhraní systému elektronické pošty, tedy ta jeho část, se kterou uživatel bezprostředně pracuje a jejímž prostřednictvím čte došlé zprávy, odpovídá na ně, připravuje nové zprávy a zadává je k odeslání, již není považována za součást aplikační vrstvy, neboť se může v každém konkrétním uzlu dosti výrazně lišit - např. ve způsobu svého ovládání (řádkovými příkazy či pomocí různých menu, s okénky či bez nich apod.). Jiným názorným příkladem může být emulace terminálů, potřebná např. pro vzdálené přihlašování (remote login). Ve světě dnes existuje nepřeberné

množství různých terminálů, a realizovat potřebné přizpůsobení mezi libovolnými dvěma typy terminálů je v podstatě nemožné. Proto se zavádí jediný "referenční" terminál - tzv. virtuální terminál - a pro každý konkrétní typ terminálu se pak vytvoří jen jediné přizpůsobení mezi tímto virtuálním terminálem a terminálem skutečným. Prostředky pro práci s virtuálním terminálem přitom jsou součástí aplikační vrstvy (neboť jsou všude stejné), zatímco prostředky pro jeho přizpůsobení konkrétnímu terminálu již součástí aplikační vrstvy nejsou.

26. Referenční model ISO/OSI - přenos dat

Ve 23. dílu našeho seriálu jsme si naznačili základní myšlenku, na které je založena funkce všech vrstvových modelů: že každá vrstva využívá služeb vrstvy bezprostředně nižší, a sama nabízí své služby vrstvě bezprostředně vyšší. Upřesněme si nyní tuto obecnou představu právě na příkladu referenčního modelu ISO/OSI a jeho terminologie a ukažme si, jak se projevuje při přenosu dat.

Ten, kdo v určité vrstvě ISO/OSI modelu něco aktivního vykonává, je označován jako **entita (entity)**. Bývá to nejčastěji objekt programové povahy (např. určitý proces), v nejnižších vrstvách to však může být i hardwarový celek (např. V/V řadič apod.). Na úrovni aplikační vrstvy jde o **aplikační entity (application entities)**, na úrovni prezentační vrstvy o **prezentační entity (presentation entities**) atd. Entity na stejné úrovni, resp. ve stejnolehlých vrstvách se v angličtině označují přívlastkem **peer** (tj. jako **peer entities**).

Entity ve vrstvě N implementují služby, které jsou využívány vrstvou N+1, resp. entitami této vrstvy. Vrstva N zde tedy vystupuje jako **poskytovatel služby (service provider)**, zatímco vrstva N+1 je v roli **uživatele služby (service user)**. Vrstva N je však současně i v roli uživatele služeb vůči vrstvě N-1, neboť využívá její služby.

Bezprostředními poskytovateli a uživateli služeb však nejsou vrstvy jako takové, ale jejich entity. Každá entita, která chce nějaké služby využívat, však musí vědět, kam se má pro ně obrátit. Musí tedy existovat jednoznačně identifikovatelná místa v rámci rozhraní mezi jednotlivými vrstvami, jejichž prostřednictvím jsou služby poskytovány, resp. využívány. V terminologii ISO/OSI modelu se tato místa nazývají **body poskytování služby (Service Access Points, zkratkou SAP)** a jsou opatřena jednoznačnými adresami. Pro snazší představu je dobré využít analogii s telefonní sítí: bod poskytování služby neboli SAP je analogií zásuvky běžné telefonní přípojky, do které lze zapojit telefonní přístroj (odpovídající entitě vrstvy N+1). Chceme-li se pak dovolat na tento telefonní přístroj (poskytnout službu entitě vrstvy N+1), musíme znát příslušné telefonní číslo (adresu bodu SAP).

Každý bod SAP slouží vždy vzájemné komunikaci právě dvou entit ze sousedních vrstev. Nemůže být tedy sdílen více entitami. Každá entita však může poskytovat své služby více entitám prostřednictvím více bodů SAP a stejně tak každá entita může využívat služby více různých entit přes více bodů SAP. Situaci ilustruje obrázek 26.1.

.cp9

.PI OBR26_1.TIF, 20, 40, 10

Obr. 26.1.: Představa entit a bodů poskytování služby (SAP)

Každá entita, která chce využívat službu jiné entity bezprostředně nižší vrstvy, musí znát nejen potřebný bod SAP, přes který je jí tato služba nabízena, ale musí také znát přesný způsob volání příslušné služby. Ten je samozřejmě pro různé služby různý, vždy však vyžaduje, aby volající předal volanému určitý objem řídicích informací, které jeho požadavek specifikují. V terminologii ISO/OSI modelu je tatořídicí informace označována jako **Interface Control Information (ICI)**. Součástí požadavku pak bývají ještě různá "užitečná" data, označovaná jako **Service Data Unit (SDU)**. Volající entita tedy v obecném případě předává volané entitě přes příslušný bod SAP na rozhraní mezi nimi dvě skupiny informací (tj. ICI a SDU), které dohromady tvoří tzv. **datovou jednotku rozhraní (Interface Data Unit,** zkratkou **IDU)** - viz obr. 26.2.

Entita, která je poskytovatelem služby a která je příjemcem jednotky IDU, si převezme řídicí informace obsažené v IDU (tj. část ICI). Z ní se dozví, co je od ní požadováno, a podle toho dále naloží s datovou částí (části SDU).

Jedním z nejčastějších požadavků entity ve vrstvě N+1 je žádost o přenos dat stejnolehlé entitě (tj. entitě vrstvy N+1) na jiném uzlovém počítači. Entita vrstvy N, která je o tuto službu požádána, nalezne příslušná data v části SDU přijaté jednotky IDU a snaží se je předat své partnerské entitě na příslušném uzlovém počítači. S touto stejnolehlou entitou komunikuje podle určitého protokolu (viz 23. díl našeho seriálu).

.PI OBR26_2.TIF, 20, 40, 10

Obr. 26.2.: Představa předávání dat mezi vrstvami

Tento protokol mimo jiné určuje, jak velké "kusy" dat si mohou obě vrstvy posílat a jakými dodatečnými informacemi řídicí povahy musí být tato užitačná data doplněna. Odesílací entita vrstvy N proto rozdělí data z jednotky SDU na tak velké části, jaké jí protokol umožňuje přenášet a připojí k nim potřebné řídicí informace. Tím vzniká tzv. **protokolární datová jednotka (PDU, Protocol Data Unit)**, tvořená kromě užitečných dat (tj. složky SDU) také složkou **PCI (Protocol Control Information)**, který obsahuje informace řídicí povahy, předepsané použitým přenosovým protokolem - viz opět obrázek 26.2.

Partnerské entity stejnolehlých vrstev si tedy vzájemně zasílají jednotky PDU (ve formě zpráv, paketů, resp. rámců). Ve skutečnosti to ale doopravdy dělají jen entity fyzické vrstvy, pro které jsou celými jednotkami PDU jednotlivé bity, a které jsou schopné si je navzájem skutečně předávat. Všechny ostatní (tj. vyšší) vrstvy si jednotky PDU vyměňují prostřednictvím entit bezprostředně nižších vrstev - při odesílání se z celé jednotky PDU stávají "užitečná" data (složka SDU). K nim se přidá potřebná řídicí informace (složka ICI) a tím vzniká datová jednotka rozhraní (jednotka IDU), která je přes bod SAP předána entitě bezprostředně nižší vrstvy ...a vše se opakuje až do úrovně fyzické vrstvy.

.cp20

Průchod uživatelských dat všemi vrstvami ISO/OSI modelu ukazuje obrázek 26.3. Z něj je názorně vidět, jak si každá vrstva na straně odesilatele v závislosti na použitém protokolu přidává k užitečným datům (jednotkám SDU) své řídicí informace (složky

PCI) - obvykle ve formě hlavičky, kterou si pak stejnolehlá partnerská vrstva na straně příjemce zase odebírá.

.PI OBR26_3.TIF, 20, 40, 10

Obr. 26.3.: Průchod přenášených dat vrstvami ISO/OSI modelu

27/ Referenční model ISO/OSI - druhy služeb

Na referenčním modelu ISO/OSI se silně projevila skutečnost, že při jeho koncipování měli hlavní slovo lidé spíše "od spojů", než "od počítačů". Celý model je totiž prodchnut spojařskou mentalitou a pohledem na svět, což se projevilo například ve výhradní podpoře tzv. spojovaných služeb na úkor služeb nespojovaných, a v představě o způsobu realizace těchto služeb.

Služby, které poskytují vrstvy různých úrovní, mohou mít charakter spojovaných či nespojovaných služeb. **Spojovaná služba** (**Connection-oriented Service**) pracuje na obdobném principu, jako telefonní systém. Chcete-li s někým hovořit po telefonu, musíte nejprve vytočit jeho číslo, a on musí na vaše volání odpovědět zvednutím telefonu. Tím mezi vámi vzniká spojení, prostřednictvím kterého spolu komunikujete, a které na konci hovoru zaniká (položením sluchátka). Analogicky je tomu i v případě dvou entit na stejných úrovních, které spolu chtějí komunikovat - nejprve musí být mezi nimi navázáno spojení. Jakmile je toto spojení navázáno, chová se v jistém smyslu jako roura - vysílající do ní na jedné straně vkládá to, co si přeje odeslat, a příjemce si to na druhé straně ve stejném pořadí odebírá.

Nespojovanou službu (**Connectionless Service**) lze naopak přirovnat k běžné listovní poště. Ta nepočítá se zřízením spojení mezi odesilatele a příjemcem, ale místo toho považuje jednotlivé části přenášených dat (zprávy) za samostatné celky, opatřené adresou svého konečného příjemce, a doručuje je nezávisle na ostatních zprávách. Jednotlivé zprávy tedy mohou být v principu přenášeny různými cestami, takže se může i stát, že při příjmu nebude zachováno jejich správné pořadí - což se u spojované služby stát nemůže.

Spojovaný (connection-oriented) charakter mají jak telefonní sítě, tak i např. většina veřejných datových sítí a jiných telekomunikačních systémů, zatímco např. v lokálních sítích mají přenosy obvykle nespojovaný (connectionless) charakter. Původní verze referenčního modelu ISO/OSI však počítala pouze se spojovanými (connection-oriented) službami a protokoly, a zavedení nespojovaných bylo provedeno až dodatečně.

Spojované služby jsou v obecném případě výhodnější pro přenos větších objemů dat. Mají sice větší jednorázovou počáteční režii (na navázání spojení), ale na druhé straně vykazují menší režii na vlastní přenos dat. Naopak nespojované služby nemají prakticky žádnou jednorázovou režii, mají však relativně vyšší režii na vlastní přenos jednotlivých částí přenášených dat. Jsou proto naopak výhodnější pro přenos menších objemů dat (tj. kratších zpráv).

Je dobré si uvědomit, že spojovaný resp. nespojovaný charakter mohou mít služby všech vrstev od aplikační po linkovou (zatímco na úrovni fyzické vrstvy již rozlišování mezi oběma variantami ztrácí smysl). V 17. dílu našeho seriálu, ve kterém jsme se zabývali veřejnými datovými sítěmi, jsme si ukazovali příklady takovýchto

služeb na úrovni síťové vrstvy - přepojování okruhů a virtuálních okruhů (které mají spojovaný charakter) a datagramové služby (která má nespojovaný charakter). V dalších pokračováních našeho seriálu si ukážeme, že služby spojovaného a nespojovaného charakteru existují mj. i na úrovni transportní vrstvy, a že např. nespojovaná transportní služba může být implementována prostřednictvím služby síťové vrstvy, která má naopak spojovaný charakter.

Kromě rozdělování služeb na spojované a nespojované je možné i jiné rozdělení - na spolehlivé a nespolehlivé.

Spolehlivá služba (**Reliable Service**) je taková, která nikdy neztrácí žádná data. Obvykle je tato služba realizována prostřednictvím vhodného mechanismu potvrzování (kdy příjemce potvrzuje úspěšné přijetí resp. znovu žádá o vyslání dat, které byly přijaty chybně). S tím je ovšem spojena určitá režie, která nemusí být vždy žádoucí - představme si např. přenos digitalizovaného zvuku. Zde je jistě výhodnější raději občas přijmout chybná data (tj. poněkud zkreslený zvuk), než připustit výpadky, způsobované potvrzováním resp. opakovaným přenosem chybně přijatých dat.

Proto mají své opodstatnění i **nespolehlivé služby** (**Unreliable Services**), což je ovšem poněkud zavádějící označení - je vhodné chápat je spíše jako služby, které mají vysokou míru spolehlivosti, neposkytují však stoprocentní záruku úspěšnosti přenosu. Jelikož ale nepoužívají mechanismy potvrzování (spojené s určitou režií), mohou být rychlejší než obdobné "spolehlivé" služby.

Prakticky jediným možným způsobem, jak zajistit spolehlivou službu, je vhodný mechanismus potvrzování. Dělení služeb na **potvrzované** (**confirmed**) a **nepotvrzované** (**unconfirmed**) proto obvykle splývá s jejich dělením na "spolehlivé" resp. "nespolehlivé".

Jak jsme si již uvedli dříve, samotný referenční model ISO/OSI neobsahuje přesné vymezení služeb, které jednotlivé vrstvy poskytují vrstvám bezprostředně nadřízeným. Zavádí pouze dosti obecnou představu o tom, jakým způsobem by mělo být zprostředkováno využití těchto služeb na rozhraní mezi jednotlivými vrstvami - prostřednictvím čtyř druhů tzv. **služebních primitiv** (**service primitives**) - viz tabulka 27.1**.** Ty je možné si představit jako požadavky na provedení určitých konkrétních akcí, nebo naopak jako signalizaci určitých situací, hodných zřetele.

Druh primitiva	Význam
Request (žádost)	Entita požaduje na službě provedení určité činnosti
Indication (indikace)	Entita má být informována o určité události
Response (Odezva)	Entita reaguje na určitou událost
Confirm (Potvrzení)	Entita je informována o výsledku vyřízení svého požadavku

Tabulka 27.1.: Druhy služebních primitiv ISO/OSI modelu

Uvažujme jako příklad potvrzovanou službu, např. zřízení spojení mezi dvěma partnerskými entitami stejnolehlých vrstev - viz též obrázek 27.2. Poskytovatelem služby je entita bezprostředně nižší vrstvy, které musí žadatel o zřízení spojení předat svou žádost ve formě primitiva typu **požadavek** (**request**). Entitě, se kterou má být spojení navázáno, se požadavek projeví jako **indikace** (**indication**), která je dalším typem služebního primitiva. Jde-li o službu potvrzovanou, následuje **odezva** (**response**), která pak vyvolá **potvrzení** (**confirmation**).

.PI OBR27_2.TIF, 20, 40, 10

Obr. 27.2.: Služební primitiva - "prostorová" představa

9

Zamysleme se ale ještě jednou nad tím, co vlastně jsou ona poněkud vágní "služební primitiva", a to z pohledu programové realizace jednotlivých služeb. Jsou služební primitiva procedury, které vyšší vrstva volá a bezprostředně nižší vrstva vykonává? V případě "požadavku" a "odezvy" je to asi představa dosti vhodná, co ale v případě "indikace" a "potvrzení", které mají příjemce upozornit na určitou skutečnost, a kterou sám nemusí vůbec očekávat? Zde již představa explicitně volané procedury selhává. Vhodnější by byla spíše představa asynchronního přerušení příjemce, což ale jen velmi málo vyhovuje dnešním vyšším programovacím jazykům a je zcela ve sporu se všemi zásadami strukturovaného programování. Z čistě spojařského pohledu na svět je to ale představa velmi přirozená - např. obyčejný telefon v důsledku "indikace" začne vesele zvonit.

.PI OBR27_3.TIF, 20, 40, 10

Obr. 27.3.: Služební primitiva - průběh v čase

28/ Fyzická vrstva

Jak jsme si uvedli již dříve, úkol fyzické vrstvy v rámci referenčního modelu ISO/OSI je zdánlivě velmi jednoduchý - zajistit přenos jednotlivých bitů mezi příjemcem a odesilatelem.

V předchozích dílech našeho seriálu jsme si také naznačili, že možných prostředků pro vlastní přenos dat je celá řada - od nejrůznějších kabelů, telefonních okruhů, přes veřejné datové sítě až např. po satelitní spoje - a že tyto se více či méně liší ve způsobu přenosu, dosažitelných přenosových rychlostech, způsobu ovládání i v dalších aspektech. Je právě úkolem fyzické vrstvy, aby se přizpůsobila konkrétním přenosovým prostředkům, vytvořila potřebné rozhraní pro jejich připojení k uzlovému počítači sítě, a prostřednictvím tohoto rozhraní je také ovládala.

Fyzické vrstvy se proto týkají standardy, které definují elektrické, mechanické, funkční a procedurální vlastnosti rozhraní pro připojení různých přenosových

prostředků a zařízení (tj. kabelů, modemů apod.) - tedy elektrické parametry přenášených signálů, jejich význam a časový průběh, vzájemné návaznosti řídících a stavových signálů, zapojení konektorů, a mnoho dalších parametrů technického i procedurálního charakteru. Úkolem entit fyzické vrstvy je pak na základě těchto standardů obsluhovat přenosové prostředky, připojené k příslušným rozhraním, a jejich prostřednictvím zajišťovat přenosy jednotlivých bitů.

Způsob připojování modemů k počítačům a terminálům definuje standard **RS-232-C** (**R**ecommended **S**tandard no. **232**, revision **C**), pocházející od americké standardizační organizace EIA (viz 22. díl našeho seriálu). Ten vymezuje přesné rozhraní mezi dvěma druhy zařízení, které se v terminologii tohoto standardu označují jako DTE (Data Terminal Equipment, v překladu: KZD neboli Koncové Zařízení přenosu Dat), a DCE (Data Circuit-Terminating Equipment, v překladu: UZD neboli Ukončující Zařízení Datového okruhu - viz 8. díl našeho seriálu). Zařízením DTE je např. počítač nebo terminál, zatímco zařízením DCE je modem. Standard RS-232-C definuje nejen počet a význam jednotlivých signálů rozhraní mezi zařízeními DTE-DCE, zapojení konektorů, ale také elektrické parametry přenášených signálů (mj. napěťové úrovně, které vyjadřují jednotlivé stavy těchto signálů). Prakticky shodný standard resp. doporučení s označením **V.24** pochází od organizace CCITT (liší se jen v několika málo zřídka používaných signálech). Doporučení V.24 však na rozdíl od standardu RS-232-C nedefinuje přesné elektrické parametry přenášených signálů (v terminologii doporučení V.24: obvodů) - ty specifikuje až doplňkové doporučení **V.28.**

Ačkoli byly oba tyto prakticky ekvivalentní standardy vyvinuty právě pro připojování modemů k počítačům a terminálů, používají se dnes i k jiným účelům, např. pro připojování tiskáren či jiných periferií (se sériovým rozhraním) k počítačům, a také ke vzájemnému propojování počítačů (tj. k propojování dvou zařízení DTE). V našem seriálu se budeme oběma těmito standardy ještě zabývat podrobněji.

Nevýhodou standardů RS-232-C a V.24 (+ V.28) je maximální dosah přibližně 15 metrů, a maximální přenosová rychlost do 20 kbit/sekundu. Většího dosahu a vyšších přenosových rychlostí lze obecně dosáhnout jen s elektrickými signály jiných parametrů. Proto byly vytvořeny další standardy (doporučení CCITT) V.10 a V.11, které jsou alternativou k doporučení V.28 (tj. definují právě a pouze elektrické parametry přenášených signálů). Předpokládají použití výkonnějších budičů a přijímačů (po řadě nesymetrických resp. symetrických, tj. takových, které přenášený logický signál vyjadřují napětím mezi jedním signálovým vodičem a zemí, resp. rozdílem napětí mezi dvěma signálovými vodiči), a umožňují dosahovat přenosové rychlosti až 10 Mbit/sekundu (viz tabulka 28.1). Doporučení V.10 a V.11 mají své ekvivalenty v doporučeních CCITT X.26 resp. X.27, a také v doporučeních EIA (konkrétně RS-423-A resp. RS-422-A).

Standard	Doporučení	Maximální	Logická 1	Logická 0
EIA	CCITT	Obvody	dosah	(u vysílače)	(u vysílače)
RS-232-C	V.28	nesymetrické	20 kbit/sekundu do 15 m	-5 až -15 V	+5 až +15V

RS-423-A V.10(X.26) nesymetrické 100 kbit/sekundu do 10 m -4 až -6 V +4 až +6 V

1 kbit/sekundu do 1000 m

RS-422-A V.11(X.27) symetrické 10 Mbit/sekundu do 10 m -2 až -6 V +2 až +6 V

100 kbit/sekundu do 1000 m

Tabulka 28.1.: Elektrické parametry a dosah signálů

Standard RS-232-C je již poměrně starého data (byl poprvé publikován v roce 1969). Koncem 70. let se jej organizace EIA rozhodla nahradit novějším standardem **RS-449**. Ten zavádí některé nové signály (především pro potřeby testování) a předpokládá použití jiných konektorů, na rozdíl od svého předchůdce však již nedefinuje elektrické charakteristiky přenášených signálů - pro ně přejímá standardy RS-423-A a RS-422-A.

Nový standard RS-449 se však přes velkou snahu organizace EIA v praxi neujal (zřejmě pro přílišnou komplikovanost oproti svému předchůdci). Proto EIA počátkem roku 1987 přišla s novou verzí (revizí D) standardu RS-232, nyní tedy s označení RS-232-D, která oproti verzi RS-232-C přidává dva nové signály (pro testování modemů).

Způsob připojování počítačů k digitálním okruhům a k veřejným datovým sítím definuje doporučení **X.21** CCITT z roku 1976. To sice zasahuje až do síťové vrstvy (neboť obsahuje popis řízení činnosti v síti s přepojováním okruhů), na úrovni fyzické vrstvy však definuje počet a význam přenášených signálů i zapojení konektorů. Nedefinuje však již elektrické vlastnosti přenášených signálů - k tomuto účelu přejímá doporučení V.10/X.26 a V.11/X.27. Kromě doporučení X.21 existuje i alternativní doporučení **X.21(bis)**, které je podmnožinou standardů RS-232-C/V.24, definuje způsob připojování k analogovým přenosovým sítím, a je zamýšleno jako dočasný standard pro dobu, kdy digitální sítě ještě nejsou všeobecně rozšířené.

Fyzickou vrstvu pokrývají také standardy řady 802, pocházející od sdružení IEEE, které jsou určené pro lokální sítě (a my se jimi samozřejmě budeme zabývat podrobněji). Nejzajímavější je z tohoto pohledu zřejmě standard IEEE 802.3 pro sběrnicové lokální sítě s přístupovou metodou CSMA/CD (označované poněkud nepřesně jako sítě typu Ethernet). Tento standard totiž ve své původní verzi (z roku 1985) předpokládal pouze použití tzv. tlustého koaxiálního kabelu (Thick Ethernet, viz 19. díl našeho seriálu). Poměrně brzy však bylo přijato několik doplňků standardu IEEE 802.3, které umožnily využít i jiná přenosová média - od tzv. tenkého Ethernetu, přes koaxiální kabel s přenosem v přeloženém pásmu, až po běžnou kroucenou dvoulinku a optická vlákna. Jednotlivé doplňkové standardy (včetně původního) pak dostaly i výstižná označení, která shrnuje tabulka 28.2. Obdobný vývoj, jako u IEEE 802.3, lze pozorovat i u ostatním standardů řady 802.

Typ Max. přenosová Přenosové Topologie Druh přenosu Max. délka

rychlost	médium	sítě	jednoho segmentu
10BASE5	10 Mbit/sekundu	"tlustý Ethernet"	sběrnice v základním pásmu do 500 m
10BASE2	10 Mbit/sekundu	"tenký Ethernet"	sběrnice v základním pásmu do 200 m
10BROAD36	10 Mbit/sekundu	koaxiální kabel 75Ů	sběrnice v přeneseném pásmu do 3600 m
1BASE5	1 Mbit/sekundu	kroucená dvoulinka	hvězda v základním pásmu do 500 m
10BASE-T	10 Mbit/sekundu	kroucená dvoulinka	hvězda v základním pásmu do 100 m
10BASE-F	10 Mbit/sekundu	optická vlákna	hvězda

Tabulka 28.2.: Přehled standardů IEEE 802.3

Vlastní specifické řešení na úrovni fyzické vrstvy vyžadují také některé novější přenosové technologie a prostředky. Fyzickou vrstvu proto pokrývají i standardy resp. doporučení, které použití těchto prostředků resp. technologií definují, například:

- pro optické přenosy doporučení ISO 9314, které definuje standard FDDI (Fiber Distributed Data Interface),

- pro sítě ISDN doporučení CCITT I.430, (které se týká výhradně fyzické vrstvy)

29/ Linková vrstva - I.

Fyzická vrstva ISO/OSI modelu zajišťuje přenos jednotlivých bitů mezi dvěma uzlovými počítači, mezi kterými existuje přímé spojení (tj. vhodný komunikační kanál resp. okruh). Linková vrstva pak využívá těchto prostředků pro přenos větších bloků dat, označovaných jako **rámce** (**frames**), a přenos těchto rámců pak sama nabízí jako svou službu bezprostředně vyšší vrstvě - vrstvě síťové.

Fyzická vrstva však nijak nerozlišuje jednotlivé bity, které přenáší. Je proto na linkové vrstvě, aby sama zajistila jejich správnou interpretaci - aby dokázala rozpoznat, které bity resp. skupiny bitů představují řídící informace (např. signalizují začátek či konec bloku, udávají jeho délku apod.), a které bity představují vlastní "užitečná" data.

V případě asynchronního přenosu - viz 2. díl našeho seriálu - jsou přenášená data členěna na znaky (stejně velké skupiny bitů). Tyto jsou pak pro potřeby přenosu "obaleny" tzv. start a stop bity, které příjemci umožňují správně rozpoznat začátek a

konec znaku. Potřebujeme-li pak přenášet data, tvořená posloupnostmi běžných (tj. tisknutelných) ASCII znaků, je nejjednodušší metodou vložit celý blok znaků mezi dvojici speciálních (tj. netisknutelných znaků): STX (Start of TeXt, začátek textu) a ETX (End of TeXt, konec textu), které jsou pro svou funkci označovány jako **řídící znaky přenosu** (**transmission control characters**). Tím se dosáhne potřebné synchronizace na úrovni rámců (**frame synchronization**), neboť znaky STX a ETX umožňují příjemci správně rozpoznat začátek a konec rámce (zatímco jednotlivé znaky rámce rozpoznává při asynchronním přenosu díky zmíněným start a stop bitům) - viz obr. 29.1. a/.

Právě naznačený způsob ovšem nelze použít v případě, kdy potřebujeme přenést (jako data) i některé řídící znaky, nebo v případě, kdy místo znaků přenášíme obecná binární data. Ta sice můžeme rozdělit na stejně velké skupiny bitů a chápat je jako kódy jednotlivých znaků, stále se nám však může stát, že nám takto vyjde právě některý ze zmíněných řídících znaků. Pak je nutné zajistit tzv. **transparenci dat** (**data transparency**), tedy umožnit, aby mezi přenášenými daty mohly být i řídící znaky, a tyto nebyly interpretovány jako řídící, ale jako "užitečná" data. Používá se k tomu technika tzv. **vkládání znaků** (**character stuffing**), kdy je před řídící znaky STX a ETX vložen ještě jiný řídící znak - znak **DLE** (Data Link Escape, změna významu následujícího znaku). Ten se ovšem může vyskytovat i mezi vlastními daty, a proto se zde každý jeho výskyt zdvojuje - viz obr. 29.1. Příjemce se pak vždy po přijetí znaku DLE rozhoduje podle následujícího znaku - je-li jím další znak DLE, jednoduše jej vypustí (a přijme předchozí znak DLE jako "užitečná" data). Je-li naopak dalším znakem znak ETX, příjemce si z toho odvodí, že se dostal na konec bloku (viz obr. 29.1. b/.

.PI OBR29_1.TIF, 20, 40, 10

Obr. 29.1. Synchronizace na úrovni rámců při asynchronním přenosu

a/ pro textová data

b/ pro binární data

Jednou z nevýhod asynchronního způsobu přenosu je vkládání nezbytných start a stop bitů, čímž se dosti podstatně snižuje efektivní přenosová rychlost. Pro vyšší přenosové rychlosti se proto používá spíše přenos synchronní (viz 2. díl seriálu).

Synchronní přenos si lze představit jako spojitý proud bitů (bez start a stop bitů), ve kterém příjemce musí správně rozpoznávat hranice mezi jednotlivými znaky (což představuje synchronizaci na úrovni znaků, **character synchronization**). Té se dosahuje pomocí speciálních znaků **SYN** (viz opět 2. díl našeho seriálu), které uvozují každý synchronně přenášený blok znaků, viz obr. 29.2.

Synchronizace na úrovni rámců (tj. správné rozpoznání začátku a konce rámce) se při synchronním přenosu může dosahovat stejně, jako při přenosu asynchronním - pomocí řídících znaků přenosu. Pak jde o tzv. **znakově orientovaný přenos** (**character-oriented transmission**).

Vkládání celých řídících znaků do přenášených dat a jejich nezbytné zdvojování při přenosu binárních dat ale opět přináší snížení efektivní přenosové rychlosti. Proto se dnes stále více uplatňuje **bitově orientovaný přenos** (**bit-oriented transmission**). Je

založen na myšlence indikovat začátek a konec rámců nikoli řídícím znakem, ale skupinou bitů. Označení "bitově orientovaný" přitom zdůrazňuje skutečnost, že přenášená data jsou vyhodnocována bit po bitu, dokud není nalezena hledaná skupina bitů, indikující začátek rámce resp. jeho konec. Počet bitů, které tvoří vlastní obsah rámce, pak nemusí nutně být násobkem osmi.

.PI OBR29_2.TIF, 20, 40, 10

Obr. 29.2.: Synchronní znakově orientovaný přenos

a/ formát rámce

b/ synchronizace na úrovni znaků

Jednou z možností pro bitově orientovaný přenos je použít stejnou skupinu bitů, tzv. **křídlovou značku** (**flag**) pro uvození i zakončení rámce - viz obr. 29.3. a/. Tato křídlová značka se pak ovšem nesmí vyskytovat "uvnitř" vlastního rámce. Obvykle je křídlová značka tvořena posloupností "01111110", a potřebná transparence dat se zajišťuje **vkládáním bitů** (**bit stuffing**), při kterém je za každých pět po sobě jdoucích jedničkových datových bitů automaticky vložen jeden nulový bit (který příjemce zase automaticky odstraňuje) - viz obr. 29.3. b/.

Další možností je uvození celého rámce (po tzv. preambuli neboli synchronizačním poli) tzv. **příznakem začátku rámce** (**start-of-frame delimiter**), za kterým následuje hlavička (header) předem stanoveného formátu, a údaj o délce rámce - viz obr. 29.3. c/. Tato varianta se používá především u lokálních sítí.

.PI OBR29_3.TIF, 20, 40, 10

Obr. 29.3.: Synchronní bitově orientovaný přenos

a/ formát rámce s křídlovou značkou

b/ představa vkládání bitů

c/ formát rámce s příznakem začátku a zadanou délkou

Podle používaného způsobu přenosu lze protokoly, používané na úrovni linkové vrstvy ISO/OSI modelu, rozdělit do dvou velkých skupin: na **znakově orientované protokoly** (**character-oriented protocols**) a **bitově orientované protokoly** (**bit-oriented protocols**).

Mezi znakově orientované protokoly patří především protokol, vyvinutý firmou IBM pod označením **Binary Synchronous Protocol**, zkráceně nazývaný **Bisync**, či jen **BSC**. V poslední době se však stále více prosazují spíše bitově orientované linkové protokoly. Nejvýznamnějším představitelem této skupiny je opět protokol, vyvinutý firmou IBM pro její síťovou architekturu SNA - protokol **SDLC** (**Synchronous Data Link Control**). Od tohoto protokolu jsou pak odvozeny téměř všechny ostatní používané bitově orientované protokoly - **HDLC** (**High-Level Data Link Control**), pocházející od organizace ISO, **LAP** (**Link Access Procedure**) a jeho různé varianty

od CCITT, či **ADCCP** (**Advanced Data Communications Control Procedure**), který je standardem ANSI.

30/ Linková vrstva - II.

Správné rozpoznání začátku a konce každého rámce i jeho jednotlivých částí není zdaleka jediným úkolem, který řeší linková vrstva referenčního ISO/OSI modelu.

Služby, které linková vrstva poskytuje vrstvě síťové, mohou mít charakter spolehlivých i nespolehlivých služeb (viz 27. díl našeho seriálu). Pro realizaci spolehlivých služeb pak linková vrstva musí mít k dispozici mechanismy pro zajištění toho, že příjemce skutečně přijme všechny vyslané rámce, a to bez chyb (v případě spojované služby ještě ve správném pořadí).

Možné způsoby, jak tento požadavek zajistit, závisí na charakteru spojení mezi vysílajícím a příjemcem. Záleží na tom, zda toto spojení je svou povahou **simplexní** (**simplex**), tj. umožňující pouze jednosměrný přenos od vysílajícího k příjemci, nebo tzv. **poloduplexní** (**half-duplex**), umožňující sice obousměrný přenos, ale nikoli současně, nebo **plně duplexní** (**full duplex**), umožňující současný přenos oběma směry.

Simplexní spoje neumožňují vytvořit zpětnou vazbu mezi vysílajícím a příjemcem. Příjemce pak nemá možnost vyžádat si nové vyslání těch rámců, které přijal jako poškozené, a se všemi případnými chybami se musí vyrovnat sám. Vysílající mu v tom může pomoci tím, že použije vhodný samoopravný kód (viz 3. díl našeho seriálu), díky kterému pak příjemce dokáže některé chyby v přenesených datech opravit sám. Tyto kódy jsou však spojeny se značnou redundancí, kvůli které výrazně klesá efektivní přenosová rychlost "užitečných" dat. Stejná situace nastává také u takových spojů, které sice nejsou simplexní, ale pracují s tak dlouhými dobami přenosu, že se na úrovni linkové vrstvy nevyplatí čekat na zpětnou vazbu od příjemce dat (příkladem mohou být družicové spoje, viz 18. díl našeho seriálu).

V případě poloduplexních a plně duplexních spojů je možné vystačit již jen se zabezpečením přenášených dat pomocí detekčních kódů (viz 3. díl seriálu). Z nich jsou nejúčinnější tzv. cyklické kódy (viz opět 3. díl našeho seriálu), které lze použít k zabezpečení rámce jako celku. Při odesílání se k obsahu rámce přidá krátký zabezpečovací údaj (typicky v rozsahu 16 bitů), a příjemce je pak na základě tohoto zabezpečovacího údaje schopen se značnou pravděpodobností rozpoznat, zda přijal rámec bez chyby, či nikoli. V druhém případě pak může využít zpětné vazby, kterou mu nabízí poloduplexní a duplexní spojení s vysílajícím, a vyžádat si na něm nové vyslání celého chybně přijatého rámce.

Právě naznačený mechanismus je obvykle implementován v podobě tzv. **potvrzování** (**acknowledgement**), přesněji: **potvrzovací zpětné vazby**, která předpokládá, že příjemce zkontroluje bezchybovost každého přijatého rámce, a o výsledku informuje vysílajícího. V angličtině je pak tato technika označována také jako **ARQ** (**Automatic Retransmission reQuest**).

Možností realizace mechanismu potvrzování existuje celářada. V prvním přiblížení je lze rozdělit na dvě velké skupiny, na tzv. **jednotlivé potvrzování** (**idle RQ, stop&wait RQ**), a **kontinuální potvrzování** (**continuous RQ**). V případě jednotlivého potvrzování vysílající odešle rámec, a pak čeká na reakci příjemce. Další rámec pak vyšle teprve poté, kdy mu příjemce signalizuje úspěšné přijetí původního

rámce. V opačném případě (kdy je mu signalizováno neúspěšné přijetí, nebo nedostane-li do určitého časového limitu žádnou odpověď), vyšle původní rámec znovu. Konkrétní implementovaný mechanismus pak může vycházet z toho, že příjemce potvrzuje pouze bezchybně přijaté rámce (zatímco přijetí chybných rámců nesignalizuje vůbec a tyto rámce jednoduše ignoruje), nebo naopak z toho, že příjemce vysílá jen **záporná potvrzení** (**negative acknowledgements**), resp. **odmítnutí** (**rejections**), která signalizují přijetí rámce s chybami a explicitně žádají o jeho opětovné vyslání. Nejefektivnější je ovšem taková varianta, při které příjemce explicitně signalizuje obě možné situace - pomocí kladného i záporného potvrzení, viz obr. 30.1.

.PI OBR30_1.TIF, 20, 40, 10

Obr. 30.1.: Jednotlivé potvrzování

Obecnou nevýhodou všech variant jednotlivého potvrzování je nutnost čekat před odesláním dalšího rámce na reakci protistrany. V případě delších dob přenosu tak mohou vznikat neúnosně velké časové ztráty, které minimalizuje až potvrzování kontinuální. To je založené na myšlence, že vysílající bude vysílat nové rámce bez toho, že by si byl jist úspěšným přijetím předchozích rámců. Po odeslání určitého rámce tudíž vysílající nečeká na zprávu o úspěšném či neúspěšném přijetí rámce, ale může ihned pokračovat vysláním dalšího rámce. Kladná resp. záporná potvrzení jednotlivých rámců pak dostává s určitým zpožděním, a reaguje na ně samozřejmě až v okamžiku, kdy je skutečně dostane. Zde je opět několik možných variant: vysílající může znovu vyslat jen ten rámec, o kterém se dozvěděl, že nebyl úspěšně přenesen (pak jde o tzv. **selektivní opakování, selective repeat**, viz obr. 30.2.), nebo znovu vyslat chybně přenesený rámec, a po něm znovu i všechny následující rámce, které již mezitím mohly být také odvysílány (pak jde o tzv. **opakování s návratem,** v angličtině o tzv. **Go-back-N** techniku, viz obr. 30.3.).

.PI OBR30_2.TIF, 20, 40, 10

Obr. 30.2.: Kontinuální potvrzování se selektivním opakováním

Opakování s návratem je obecně méně efektivní než selektivní opakování, neboť může způsobit opakované vyslání rámců, které již byly přijaty bezchybně. Selektivní opakování zase vyžaduje ke své implementaci poměrně velkou kapacitu vyrovnávacích pamětí, a to jak na straně příjemce, tak i na straně vysílajícího.

.PI OBR30_3.TIF, 20, 40, 10

Obr. 30.3. Kontinuální potvrzování s návratem

Při našich úvahách o mechanismech potvrzování je dobré se zamyslet také nad tím, jakou konkrétní podobu mají kladná i záporná potvrzení, která příjemce datových

rámců vrací jejich odesilateli. Mohou to být samostatné rámce zvláštního typu, které nenesou žádná "užitečná" data, ale mají pouze řídící charakter. Pak jde o tzv. **samostatné potvrzování**, které ovšem svými řídícími rámci zatěžuje přenosové cesty na úkor datových rámců s "užitečnými" daty. Efektivnější alternativou, použitelnou však jen v případě obousměrného toku "užitečných" dat, je vkládat potvrzující informace přímo do datových rámců - této technice se v angličtině říká **piggybacking.**

Dalším, velmi významným úkolem linkové vrstvy ISO/OSI modelu je zajistit, aby vysílající svými daty nezahltil příjemce. Linková vrstva se tedy musí zabývat také tzv. **řízením toku** (**flow control**), které má zajistit, aby vysílající skutečně vysílal jen tehdy, kdy je přijímající vůbec schopen nějaká data přijímat - a ne např. tehdy, když příjemce nemá k dispozici dostatečně velký objem vyrovnávací paměti pro uložení přijímaných dat, nebo je zaneprázdněn jinou činností (obsluhou naléhavého přerušení, zpracováním dříve přijatých dat apod.).

Určujícími jsou přitom možnosti příjemce - ten musí mít možnost dočasně pozastavit vysílaní dat, a později je zase obnovit. Na úrovni linkové vrstvy je nejjednodušší dočasně pozastavovat vysílání celých rámců. V případě jednotlivého potvrzování (viz výše) k tomu příjemci stačí nepotvrdit posledně přijatý rámec (nebo na něj reagovat záporným potvrzením). V případě kontinuálního potvrzování není situace o nic složitější - vysílající zde totiž vysílá "dopředu" (tj. bez potvrzení) vždy jen určitý maximální počet rámců. V souladu s obrázkem 30.4. tak vzniká "okénko" již vyslaných ale dosud nepotvrzených rámců, které prostřednictvím svých potvrzení posouvá právě příjemce rámců. Díky tomu má i tomto případě možnost podle svých potřeb dočasně pozastavit vysílání (tj. posun okénka) a později jej zase obnovit.

Kvůli charakteristické představě zmíněného "okénka" se metoda kontinuálního potvrzování často označuje také jako **metoda okénka** (**sliding window method**).

.PI OBR30_4.TIF, 20, 40, 10

Obr. 30.4. Představa metody okénka (sliding window)

31/ Linková vrstva - III.

V minulých dvou dílech našeho seriálu jsme se zabývali postavením a funkcí linkové vrstvy v rámci referenčního modelu ISO/OSI. Naznačili jsme si, že linková vrstva musí zajišťovat potřebnou synchronizaci na úrovni jednotlivých znaků i celých rámců, a starat se o bezchybné doručení všech rámců od odesilatele k jejich příjemci. Pro činnost linkové vrstvy je však velmi důležitý také konkrétní způsob vzájemného propojení jednotlivých uzlů, mezi kterými má přenos rámců zajišťovat. Podívejme se proto nyní, jaké jsou možnosti takovéhoto propojení, a jaké důsledky vyplývají z jednotlivých alternativ.

Dva uzlové počítače mohou být propojeny pomocí **dvoubodového spoje** (**point-to-point connection**) podle obrázku 31.1. Dvoubodový spoj může být realizován např. pomocí kroucené dvoulinky, koaxiálního či optického kabelu (viz obr. 31.1 a/), nebo také prostřednictvím veřejné telefonní sítě (jako sítě s přepojováním okruhů), viz obr. 31.1 b/. Podstatná je přitom skutečnost, že mezi oběma uzly existuje (alespoň po dobu přenosu) přímý přenosový kanál, a protokol linkové vrstvy pak zajišťuje přímou komunikaci obou koncových účastníků.

.PI OBR31_1.TIF, 20, 40, 10

Obr. 31.1.: Představa dvoubodového spoje

Poněkud jiná situace však nastává v případě, kdy jsou dva uzly propojeny prostřednictvím sítě s přepojováním paketů - například pomocí veřejné datové sítě na bázi X.25 (viz 17. díl našeho seriálu). Sítě tohoto typu si totiž samy zajišťují vše, co je potřeba pro "interní" přenos paketů (tj. až do úrovně síťové vrstvy, včetně). Linková vrstva zde pak má na starosti jen přenos rámců mezi koncovým uzlem a místem jeho připojení na datovou síť - viz obrázek 31.2.

.PI OBR31_2.TIF, 20, 40, 10

Obr. 31.2.: "Dosah" protokolů linkové vrstvy v případě propojení prostřednictvím veřejné datové sítě

Další možností, typickou spíše pro propojení na kratší vzdálenosti, je použití tzv. **mnohobodového spoje** (**multipoint connection**), který vzájemně propojuje více uzlů - například podle obrázku 31.3. - a umožňuje přenos dat mezi kterýmikoli dvěma uzly. Umožňuje dokonce i přenos dat z jednoho uzlu (v roli vysílajícího) současně do více uzlů (v roli přijímajících) - pro tuto svou schopnost se v angličtině tento druh propojení označuje také jako tzv. **broadcast channel** (doslova: přenosový kanál, umožňující "rozesílání").

Pro mnohobodový spoj je však podstatná skutečnost, že jde o sdílený prostředek, který neumožňuje vícenásobné přidělení - v roli vysílajícího může být vždy nejvýše jeden uzel. Pokud tedy dojde k situaci, že o získání tohoto sdíleného prostředku (tj. práva vysílat po mnohobodovém spoji) bude usilovat více uzlů současně, musí existovat mechanismus, který mezi všemi žadateli umožní vybrat jednoho, a tomu pak prostředek přidělit (tj. nechat jej vysílat).

Možností realizace je opět více. Jednou z nich je existence centrálního arbitra, který sám rozhoduje o využití sdíleného prostředku - jednotlivým žadatelům přiděluje právo vysílat po mnohobodovém spoji. Může tak činit na základě explicitních žádostí jednotlivých žadatelů, což ovšem znamená, že pro tyto žádosti je nutné vyhradit určitou část přenosové kapacity mnohobodového spoje, nebo vytvořit další vhodné propojení mezi centrálním arbitrem a ostatními uzly pro přenos těchto žádostí. V praxi se však používá spíše tzv. **metoda výzvy** (**polling**), při které se centrální arbitr sám postupně obrací na jednotlivé potenciální žadatele a zjišťuje, zda chtějí něco vyslat.

.PI OBR31_3.TIF, 20, 40, 10

Obr. 31.3.: Představa mnohobodového spoje

Znakově i bitově orientované protokoly linkové vrstvy, o kterých jsme se zmiňovali ve 29. dílu našeho seriálu obvykle umožňují používat jak dvoubodové, tak i mnohobodové spoje. V případě mnohobodových spojů pak počítají právě s existencí

centrálního arbitra - tím je u těchto protokolů jedna ze stanic, která má postavení tzv. **řídící stanice** (**supervisory, master)**, zatímco ostatní stanice vystupují v roli **podřízených** (**slave**) **stanic**. Řídící stanice přiděluje jednotlivým podřízeným stanicím právo vysílat na základě jejich kladné reakce na zaslanou výzvu (poll). Stanice, která v určitém okamžiku získá právo vysílat, se dostává do postavení tzv. **hlavní stanice** (**primary**) a sama si volí, komu chce svá data vyslat - prostřednictvím zvláštní výzvy provede tzv. **výběr** (**selection**) jedné nebo několika dalších stanic, které budou její data přijímat, a budou tedy vůči ní vystupovat jako tzv. **vedlejší stanice** (**secondary**)

Právě naznačený mechanismus v sobě skýtá jedno potenciální nebezpečí - v případě výpadku centrálního arbitra (řídící stanice) se celá síť na bázi mnohobodového spoje stává nepoužitelnou.

Existují však i jiné alternativy, které existenci centrálního arbitra nepředpokládají. Jsou založeny na myšlence, že všichni žadatelé o právo vysílat po mnohobodovém spoji se dokáží mezi sebou dohodnout, a vybrat ze svého středu jednoho, který pak skutečně začne vysílat. Jde vlastně o jakousi soutěž, do které se musí přihlásit každá stanice, která chce získat právo vysílat pro mnohobodovém spoji (tj. stát se stanicí hlavní). Tato soutěž musí samozřejmě mít přesně stanovená pravidla, která všichni soutěžící musí dodržovat - jsou definována ve formě tzv. **přístupové metody** (**access method**).

"Decentralizovaný" způsob řízení mnohobodového spoje, založený na různých přístupových metodách, je charakteristický hlavně pro lokální sítě. Nejrozšířenější jsou dnes dvě přístupové metody: metoda **CSMA/CD**, která se používá v lokálních sítích typu Ethernet, a metoda **Token passing** (tzv. předávání pověření resp. "peška"), která se používá v sítích Token Ring a Token Bus. O obou metodách si samozřejmě budeme povídat podrobněji, pokusme se je však ještě zařadit do kontextu sedmivrstvého referenčního modelu ISO/OSI.

Přístupová metoda, zajišťující korektní přístup ke sdílenému mnohobodovému spoji, musí být implementována nad fyzickou vrstvou - neboť sama využívá služeb této vrstvy pro přenos jednotlivých bitů. Linková vrstva, která zabezpečuje přenos celých rámců, by ale již měla mít potřebný přístup ke sdílenému mnohobodovému spoji zajištěn. Přístupová metoda by tudíž měla být implementována mezi fyzickou a linkovou vrstvou - což by ovšem znamenalo "vsunutí" zcela nové vrstvy do referenčního modelu, která je navíc zapotřebí jen v případě těch lokálních sítí, které používají sdílené mnohobodové spoje bez centrálního arbitra. Celá situace se vyřešila tak, že se u těchto sítí "původní" linková vrstva rozděluje na dvě části resp. podvrstvy - nižší podvrstvu **MAC** (**Media Access Control sublayer**, **podvrstva řízení přístupu k přenosovému médiu**), ve které je implementována příslušná přístupová metoda, a vyšší podvrstvu **LLC** (**Link Layer Control**, **podvrstvu řízení logického spoje**), která zajišťuje všechno to, co jsme dosud přisuzovali linkové vrstvě jako takové - viz obrázek 31.4.

Přístupové metody pro dnes nejrozšířenější lokální sítě jsou definovány ve standardech IEEE řady 802, o kterých jsme se již zmiňovali v souvislosti s fyzickou vrstvou ISO/OSI modelu ve 28. dílu našeho seriálu. Vztah těchto standardů k ISO/OSI modelu a jejich "pokrytí" názorně ukazuje obrázek 31.4.

.PI OBR31_4.TIF, 20, 40, 10

Obr. 31.4.: Vztah standardů IEEE 802 a ISO/OSI

32/ Síťová vrstva - I.

Chtějí-li spolu komunikovat dva uzly počítačové sítě, mezi kterými neexistuje přímé spojení, je nutné pro ně najít alespoň spojení nepřímé - tedy vhodnou cestu, vedoucí přes mezilehlé uzly od jednoho koncového uzly ke druhému. Možných cest může být samozřejmě více, někdo je však musí najít, jednu z nich vybrat, a pak také zajistit správné předávání dat po této cestě. Všechny tyto úkoly má v referenčním modelu ISO/OSI na starosti síťová vrstva.

Uvažujme příklad počítačové sítě na obrázku 32.1 a/ a situaci, kdy je potřeba přenést určitá data z uzlu A do uzlu D. Zdrojem těchto dat nechť je uživatelský proces, běžící na počítači A (může to být například program pro práci s elektronickou poštu a jím generovaná data zprávou, určenou pro účastníka na uzlu resp. počítači D). Uživatelský proces na počítači A předá svá data k odeslání aplikační vrstvě, která je zase předá vrstvě prezentační atd. (viz obrázek 32.2.). Když se příslušná data dostanou až na úroveň síťové vrstvy, musí tato rozhodnout, kudy je skutečně odeslat. V našem konkrétním případě (v síti dle obrázku 32.1. a/) je toto rozhodnutí velmi jednoduché, jediná cesta "ven" zde totiž vede přes uzel E. Data, určená k doručení do uzlu D, proto síťová vrstva uzlu A předá své bezprostředně nižší (linkové) vrstvě s požadavkem na jejich odeslání do uzlu E.

.PI OBR32_1.TIF, 20, 40, 10

Obr. 32.1.: Příklad topologie sítě

Jelikož mezi uzly A a E existuje přímé spojení, dokáže lInková vrstva uzlu A předat blok dat (na úrovni linkové vrstvy označovaný jako rámec) své partnerské linkové vrstvě na uzlu E. Jak již ale víme, ve skutečnosti tak činí prostřednictvím fyzické vrstvy.

Linková vrstva na uzlu E předá přijatý rámec své bezprostředně vyšší vrstvě, tj. vrstvě síťové. Ta musí z obsahu rámce poznat, že jde o data určená k doručení do uzlu D. Na základě znalosti topologie sítě (tj. způsobu propojení jednotlivých uzlů) tato vrstva zjistí, že cesta do uzlu D vede dále buď přes uzel G, nebo přes uzel F. Musí se rozhodnout pro jednu z obou možností - předpokládejme, že se rozhodne pro cestu přes uzel G. Data, která převzala od linkové vrstvy, proto síťová vrstva uzlu E vrátí své bezprostředně nižší vrstvě s požadavkem na odeslání do uzlu G.

V uzlu G se situace opakuje. Linková vrstva předá přijatý rámec síťové vrstvě, která jej vrátí linkové vrstvě zpět s požadavkem na odeslání do uzlu D, kam již z uzlu G vede přímé spojení.

V uzlu D se data dostanou analogickým způsobem až na úroveň síťové vrstvy. Ta rozpozná, že jde o data, určená právě danému uzlu, a proto je již nevrací vrstvě linkové, ale předá je své bezprostředně vyšší (tj. transportní) vrstvě. Odtud jsou pak postupně předávána směrem k vyšším vrstvám, až se dostanou k té entitě resp. procesu, která je jejich konečným příjemce. Celý postup názorně ukazuje obrázek 32.2.

.PI OBR32_2.TIF, 20, 40, 10

Obr. 32.2.: Představa průchodu dat vrstvami

Úkoly síťové vrstvy

Pokusme se nyní, na základě výše uvedeného příkladu, o zobecnění úkolů, které v referenčním modelu ISO/OSI plní síťová vrstva.

Kdykoli transportní vrstva předává vrstvě síťové nějaká data k odeslání, připojuje k nim pouze informaci o tom, kdo má být jejich konečným příjemcem. Pro každý samostatně přenášený blok dat, který se na úrovni síťové vrstvy označuje jako **paket** (zatímco na úrovni linkové vrstvy jako rámec) pak musí síťová vrstva rozhodnout, kterým "směrem" jej má skutečně odeslat. Jakmile toto rozhodnutí učiní, předá příslušný paket vrstvě linkové spolu s údajem o zvoleném směru.

Nejdůležitějším úkolem síťové vrstvy je tedy tzv. **směrování** (**routing**), které představuje právě ono zmíněné rozhodování o směru odesílání jednotlivých paketů. Není jistě třeba zdůrazňovat, že k tomu síťová vrstva potřebuje alespoň základní informace o topologii celé sítě. Konkrétních způsobů směrování, či spíše postupů resp. algoritmů volby vhodného směru přitom existuje celá řada. Od jednoduchých statických metod, které nejsou schopny reagovat na dynamické změny v síti, až po adaptivní metody, které se dokáží přizpůsobit aktuálnímu stavu sítě, jejímu zatížení, případným výpadkům některých uzlů či spojů apod.

Metodám směrování se budeme podrobněji věnovat v dalších pokračováních našeho seriálu. Vraťme se však ještě k úkolům síťové vrstvy - kromě vlastního směrování (chápeme-li jej jen jako rozhodování o dalším směru) musí síťová vrstva zajišťovat i jeho skutečnou realizaci. Tedy v mezilehlých uzlech zajišťovat potřebné předávání jednotlivých paketů na cestě k jejich koncovému příjemci.

S tím dosti úzce souvisí i další úkol síťové vrstvy - předcházet přetížení či dokonce zahlcení částí sítě, řídit tok dat a dbát o co možná nejrovnoměrnější využití všech přenosových prostředků a kapacit.

Při vzájemném propojení dvou či více sítí pak přibývá síťové vrstvě ještě jeden důležitý úkol - zajišťovat nezbytné předávání paketů mezi jednotlivými sítěmi.

Odlišné pohledy na svět

Udělejme si nyní malé shrnutí toho, jaký pohled na celou síť a vzájemné propojení jednotlivých uzlů má transportní, síťová a linková vrstva ISO/OSI modelu - umožní nám to snáze pochopit postavení a význam síťové vrstvy: transportní vrstva v každém uzlu sítě vychází z představy, že mezi jejím uzlem a koncovým příjemcem dat existuje přímé spojení, a proto adresuje svá data přímo tomuto koncovému příjemci. Síťová vrstva si již uvědomuje skutečnou topologii sítě a ví, že představa transportní vrstvy nemusí být správná (činí však vše pro to, aby transportní vrstvě její iluzi zachovala). Pro linkovou vrstvu pak již není topologie celé sítě relevantní - té stačí znát jen ty uzly, se kterými má "její" uzel přímé spojení. Nezná dokonce ani koncového příjemce dat, obsažených v rámcích, které sama přenáší.

.PI OBR32_3.TIF, 20, 40, 10

Obr. 33.3.: "Pohled" transportní, síťové a linkové vrstvy na síť

Komunikační podsíť a hostitelské počítače

Příklad topologie sítě, uvedený na obrázku 32.1., naznačuje nejobecnější situaci, kdy všechny uzlové počítače sítě mají stejné postavení a mohou vystupovat jak v roli zdrojů a koncových příjemců dat, tak i v roli mezilehlých uzlů.

V praxi však může často docházet k tomu, že meziuzly nejsou plnohodnotnými uzly počítačové sítě, ale pouze jednoúčelovými "přepojovači", které vytváří potřebnou komunikační infrastrukturu pro propojení ostatních "plnohodnotných" uzlů. Právě s takovouto představou počítal také referenční model ISO/OSI ve své původní verzi (kdy uvažoval jen tzv. spojované služby, viz 27. díl našeho seriálu). Podle této představy je oddělena čistě komunikační funkce sítě a je svěřená tzv. **komunikační podsíti** (**communication subnet, subnetwork**), zatímco vlastní aplikační funkce zajišťují tzv. **koncové systémy** (**end systems**). S podobnou představou pracovala například i jedna z prvních rozlehlých sítí ARPANET, jejíž terminologie se nejvíce prosadila do praxe. Koncové systémy, na kterých jsou provozovány uživatelské aplikace, označuje jako **hostitelské počítače** (**host computers, hosts**), zatímco jednotlivé "přepojovací" uzly v rámci komunikační podsítě označuje jako **IMP** (**Interface Message Processor**) - viz obrázek 32.1. a/. V literatuře se však často používají také termíny **uzel přepojování paketů** (**Packet Switching Node**) či **mezilehlý systém** (**Intermediate Node**) nebo **datová ústředna** (**Data Switching Exchange**).

33/ Síťová vrstva - II.

Otázka volby mezi spojovanými a nespojovanými službami (connection-oriented vs. connectionless službami, viz 27. díl našeho seriálu) je snad nejkontroverznější právě na úrovní síťové vrstvy. Když vznikal referenční model ISO/OSI, mezi jeho tvůrci měli značnou převahu zastánci spojovaných služeb. Proto také referenční model původně počítal jen s tímto druhem síťových služeb. Zastánci druhé alternativy však alespoň dodatečně prosadili do referenčního modelu i služby spojované. V čem je ale skutečné jádro sporu mezi oběma tendencemi?

Připomeňme si nejprve, že spojovaná služba funguje obdobně jako veřejná telefonní síť - předpokládá nejprve navázání spojení mezi oběma účastníky, pak vlastní přenos dat prostřednictvím tohoto spojení, které se v jistém smyslu chová jako roura, kterou se vlastní data "protlačují", a nakonec vyžaduje ukončení (rozvázání) spojení. Naopak nespojovaná služba funguje obdobně jako běžná listovní pošta - každý jednotlivý paket (resp. datagram) doručuje samostatně a nezávisle na ostatních, bez toho, že by se navazovalo spojení mezi příjemcem a odesilatelem. Další analogií s listovní poštou je rozdíl mezi spolehlivou a nespolehlivou nespojovanou službou - spolehlivá varianta je obdobou doporučené zásilky, která se nemůže (alespoň teoreticky) ztratit, zatímco nespolehlivá verze je obdobou obyčejné zásilky, u které pošta negarantuje její doručení, a která se může beze stopy ztratit.

Zastánci spojovaných služeb, rekrutující se především z kruhů spojových organizací, zastávají názor, že uživatelé resp. vyšší vrstvy potřebují maximálně spolehlivou a jednoduše použitelnou službu pro přenos dat, která by je zbavila všech starostí se zabezpečením vlastního přenosu dat (např. s potřebným řízením toku, opravou chyb, zajištěním správného pořadí jednotlivých doručovaných paketů atd.). Tedy spolehlivou spojovanou síťovou službu pro přenos paketů.

Druhá strana, reprezentovaná především lidmi kolem sítě Internet, argumentuje svou dlouhodobou zkušeností s provozováním velké počítačové sítě v reálných

podmínkách. Podle ní je nutné považovat komunikační infrastrukturu (podsť, viz minulý díl našeho seriálu) za nespolehlivou, bez ohledu na to, jak je navržena (tedy i v případě, že se sama snaží být spolehlivou). Koncové systémy (hostitelské počítače, viz minule) musí počítat s tím, že pakety se mohou v podsíti ztrácet, a vše potřebné k zajištění spolehlivosti si tudíž musí realizovat sami - tedy detekci a opravu chyb, řízení toku atd. To ovšem vede na požadavek používat na úrovni síťové vrstvy jen ty nejrychlejší a nejjednodušší služby (na úrovni operací typu "vyšli paket" a "přijmi paket"), a spolehlivost zajistit až v bezprostředně vyšší, transportní vrstvě. Tato druhá strana tedy požaduje na úrovni síťové vrstvy jen jednoduchou nespolehlivou nespojovanou službu.

Na celý spor mezi oběma tábory je možné se dívat i z poněkud jiného úhlu. Spojové organizace jako zastánci spojovaných služeb chtějí poskytovat uživatelům co nejkomplexnější a nejúplnější služby - je to asi přirozené, neboť právě za ně jsou placeny. Druhá strana naopak tvrdí, že výpočetní kapacita je dnes tak laciná, že není nejmenší problém, aby si co nejvíce funkcí zajišťovaly až jednotlivé hostitelské počítače (obvykle ve vlastnictví uživatelů). Dalším silným argumentem této skupiny je pak to, že charakter některých síťových aplikací (jako např. přenos digitalizovaného zvuku a sběr dat v reálném čase) upřednostňuje rychlý přenos před přenosem spolehlivým.

Rozpor mezi spojovanými a nespojovanými službami na úrovni síťové vrstvy je tedy ve své podstatě sporem o to, kam umístit veškerou složitost - tedy funkce, spojené se zajištěním spolehlivosti, které tak jako tak musí být někde realizovány. Zastánci spojovaných služeb je chtějí umístit do síťové vrstvy, zatímco zastánci nespojovaných služeb požadují jejich umístění až do vrstvy transportní (zatímco od vrstvy síťové požadují jen nejjednodušší nespolehlivou nespojovanou službu).

CONS i CLNS

Celý spor byl nakonec vyřešen tak, že aktualizovaná verze referenčního modelu ISO/OSI již počítá na úrovni síťové vrstvy s možností poskytování obou druhů služeb - spojovaných síťových služeb (**CONS - Connection-Oriented Network Services**) i nespojovaných síťových služeb (**CLNS, ConnectionLess Network Services**).

Služby vs. jejich realizace

Druh služeb, které síťová vrstva poskytuje své bezprostředně vyšší (tj. transportní) vrstvě, nesmíme mechanicky ztotožňovat s tím, jak síťová vrstva skutečně funguje "uvnitř". Základní možnosti jsme si naznačili již v 17. dílu našeho seriálu, kdy jsme si povídali o veřejných datových sítích. Pomineme-li možnost tzv. přepojování okruhů, která vytváří přímé fyzické spojení mezi oběma koncovými účastníky, jde o dvě základní varianty: první z nich jsou tzv. **virtuální okruhy** (též: **virtuální spoje, virtual calls, virtual circuits**) jako mechanismus spojovaného charakteru, který před vlastním přenosem datových paketů předpokládá "vytyčení" logické cesty (virtuálního okruhu) mezi oběma koncovými účastníky.

Druhou alternativou je pak přenos **datagramů**, který má nespojovaný charakter, každý jednotlivý paket (nyní nazývaný **datagram**) doručuje samostatně, nezávisle na ostatních paketech, a nepředpokládá žádné vytyčení cesty od odesilatele k jejich příjemci.

Je jistě zřejmé, že spojované síťové služby (CONS) se obvykle realizují prostřednictvím komunikační podsítě, která používá mechanismus virtuálních okruhů, zatímco nespojované síťové služby (CLNS) se realizují prostřednictvím podsítě, které pracuje na bázi přenosu jednotlivých datagramů. Není to ale jediná možnost. Spojované síťové služby lze bez větších problémů poskytovat i pomocí podsítě, která pracuje s přenosem jednotlivých datagramů (jestliže se každý paket místo virtuálním okruhem ve skutečnosti přenáší samostatně jako datagram). Teoreticky je možná i opačná kombinace - nespojované služby v podsíti, která pracuje s virtuálními okruhy. Příliš efektivní to ale není - pro přenos každého jednotlivého datagramu je nutné nejprve zřídit, a pak zase zrušit potřebný virtuální okruh. Pokud je ale k dispozici jen podsíť s virtuálními okruhy, nic jiného nezbývá.

Naznačme si nyní poněkud podrobněji, jak jsou implementovány a jak fungují mechanismy virtuálních okruhů a přenosu datagramů. Pomůže nám to lépe pochopit podstatu sporu mezi zastánci spojovaných a nespojovaných služeb.

Virtuální okruhy vs. datagramy

Mechanismus virtuálních okruhů předpokládá, že mezi zdrojem dat a jejich koncovým příjemcem je v rámci navazování spojení nejprve "vytyčena" logická cesta (virtuální okruh), po které jsou pak postupně přenášeny jednotlivé pakety.

Vytyčit cestu ve skutečnosti znamená, že se právě jednou (v rámci navazování spojení) najde taková posloupnost mezilehlých uzlů (resp. uzlů IMP, viz minule), která vede od zdroje dat až k jejich koncovému příjemci, a je v daný moment považována za optimální. Údaje o této cestě (virtuálním kanálu) se pak uchovávají v jednotlivých meziuzlech jako položky speciální tabulky - viz obr. 33.1.b/.

Jednotlivé datové pakety, které jsou pak skutečně přenášeny, nejsou označeny adresou svého konečného příjemce (která může být relativně dlouhá), ale pouze označením příslušného virtuálního kanálu. Síťová vrstva v každém meziuzlu pak podle tohoto údaje zjistí ve své tabulce, kterým směrem má paket předat dále, a učiní tak.

.PI OBR33_1.TIF, 20, 40, 10

Obr. 33.1.: Představa implementace mechanismu virtuálních

okruhů

a/ topologie sítě

b/ tabulky síťové vrstvy v jednotlivých uzlech

Naproti tomu při přenosu datagramů se předpokládá, že mezi zdrojem dat a jejich koncovým příjemcem není navazováno přímé spojení, a jednotlivé datové pakety (datagramy) jsou vysílány "naslepo", v dobré víře, že jejich příjemce vůbec existuje a bude schopen je přijmout. Každý datagram je přitom doručován nezávisle na ostatních (tak jako např. běžný dopis v případě listovní pošty) - to znamená, že v každém meziuzlu musí síťová vrstva vždy znovu rozhodnout, kterým směrem má být datagram poslán dále. V jednotlivých meziuzlech si proto síťová vrstva nevytváří obdobu tabulky, popisující virtuální okruhy. Místo toho pracuje s tabulkou, ve které

má alespoň základní informace o topologii sítě (například jako na obrázku obr. 33.2.), a podle ní přijímá svá rozhodnutí o nejvhodnějším směru dalšího přenosu každého jednotlivého datagramu.

Hlavní výhodou virtuálních okruhů je skutečnost, že rozhodování o dalším směru přenosu paketů se v každém meziuzlu přijímá jen jednou, a nikoli pokaždé znovu, jako je tomu v případě datagramů. Nevýhodou je naopak statický charakter tohoto rozhodnutí (a tím i zvolené cesty resp. virtuálního okruhu), které není možné dynamicky přizpůsobovat okamžitému stavu sítě - což je naopak možné v případě přenosu datagramů.

.PI OBR33_2.TIF, 20, 40, 10

Obr. 33.2.: Představa implementace mechanismu datagramů

a/ topologie sítě

b/ tabulky síťové vrstvy v jednotlivých uzlech

Virtuální okruhy vykazují větší jednorázovou počáteční režii (na navázání spojení resp. vytvoření logického okruhu), a menší relativní režii na přenos jednotlivých paketů, zatímco v případě datagramů je tomu přesně naopak. Srovnání dalších aspektů nabízí také tabulka 33.3.

Obecně lze říci, že mechanismus virtuálních okruhů je výhodnější v případě přenosu menších paketů (jako je tomu např. při interaktivních aplikacích), zatímco varianta s přenosem datagramů je výhodnější pro menší počty relativně větších paketů, a ve verzi nespolehlivé (resp. nepotvrzované) datagramové služby také tam, kde jde především o rychlost.

	virtuální okruhy	datagramy
navazování a ukončování spojení	ano	ne
adresování	každý paket obsahuje jen číslo virtuálního okruhu	každý datagram obsahuje adresu zdroje i příjemce datagramu
směrování	cesta volena jen jednou v rámci navazování spojení,	každý datagram je směrován samostatně

a všechny pakety jsou přenášeny po této jediné cestě		
detekce a oprava chyb	realizována v síťové vrstvě	realizována v transportní vrstvě
správné pořadí paketů	zajištěno	není zajištěno
řízení toku dat mezi koncovými účastníky	ano	ne
efekt výpadku meziuzlu	zanikají všechny zřízené virtuální kruhy	ztrácí se jen právě přenášené datagramy
složitost soustředěna do:	síťové vrstvy	do transportní vrstvy
vhodné pro realizaci:	spojovaných síťových služeb (CONS),	spojovaných i nespojovaných síťových služeb (CONS i CLNS)

Tabulka 33.3.: Srovnání virtuálních okruhů a datagramů

34/ Síťová vrstva - směrování

Neexistuje-li v počítačové síti přímé spojení mezi odesilatelem dat a jejich koncovým příjemcem, je úkolem síťové vrstvy nalézt mezi nimi alespoň takovou cestu, která vede přes některé jiné uzly sítě. Z předchozích dílů našeho seriálu již víme, že v případě tzv. virtuálních okruhů se takováto cesta hledá jen jednou, a to na začátku komunikace obou koncových účastníků (v rámci navazování spojení resp. zřizování virtuálního okruhu), zatímco u tzv. datagramové služby se vhodná cesta hledá pro každý jednotlivý paket (resp. datagram) pokaždé znovu. Víme již také, že problematika hledání cest v síti se obecně označuje jako směrování (routing). Co však ještě nevíme, je jak se taková cesta vlastně hledá.

Pro nalezení vhodné cesty od odesilatele až ke koncovému příjemci existuje celá řada **algoritmů směrování** (**routing algorithms**), které jsou založeny na různých myšlenkách a principech, a vyžadují různé stupně znalosti sítě, její topologie a dalších parametrů statického i dynamického charakteru. Všechny algoritmy směrování by ovšem měly být korektní, tedy dávat jen takové výsledky, které jsou správné a použitelné, měly by být co možná nejjednodušší, nejsnáze implementovatelné, a jejich

režie by měla být minimální. Současně s tím by ale algoritmy směrování měly být i tzv. robustní, tedy schopné vyrovnat se s nepředvídanými výpadky, poruchami či jinými nestandardními situacemi. Měly by také usilovat o optimální využití celé sítě a její přenosové kapacity, a přitom nikoho nediskriminovat - tedy někomu hledat "lepší" cesty, a někomu "horší".

Výsledným efektem aplikace algoritmů směrování by mělo být to, aby síťová vrstva v každém z uzlů sítě věděla, kudy poslat dále takový paket, který není určen přímo jejímu uzlu. Konkrétní pokyny pro směrování paketů resp. datagramů, které vznikají na základě aplikace algoritmů směrování, se pak v jednotlivých uzlech uchovávají ve formě tzv. **směrovacích tabulek** (**routing tables**).

Adaptivní a neadaptivní směrování

V prvním přiblížení si můžeme rozdělit algoritmy směrování na dvě velké skupiny. Do první z nich budou patřit takové, které se snaží průběžně reagovat na skutečný stav sítě, a brát jej do úvahy při svém hledání nejvhodnější cesty. Dokáží se tedy přizpůsobit okamžitému stavu sítě, a proto se obecně označují jako **adaptivní algoritmy** (**adaptive algorithms**). Naproti tomu **neadaptivní algoritmy** (**nonadaptive algorithms**) nevyužívají žádné informace dynamického charakteru - např. údaje o okamžitém zatížení jednotlivých přenosových cest, výpadcích, délkách čekacích front v jednotlivých uzlech apod. Svá rozhodnutí staví pouze na informacích statického charakteru, které jsou předem známy. Díky tomu lze neadaptivní algoritmy použít k nalezení všech potřebných cest ještě před uvedením sítě do provozu, a potřebné informace pak jednorázově zanést do směrovacích tabulek jednotlivých uzlů sítě. Kvůli statickému charakteru výchozích údajů se použití neadaptivních algoritmů někdy označuje také jako tzv. **statické směrování** (**static routing**). Je vhodné tam, kde topologie sítě je skutečně neměnná, kde prakticky nedochází k výpadkům a kde se příliš nemění ani intenzita provozu resp. zátěž sítě.

Centralizované směrování

Adaptivní algoritmus může být koncipován tak, že veškeré informace o aktuálním stavu celé sítě se průběžně shromažďují v jediném centrálním bodě, tzv. **směrovacím centru** (**RCC, Routing Control Center**), které pak na jejich základě samo přijímá všechna potřebná rozhodnutí, a ostatním uzlům je oznamuje. Pak jde o tzv. **centralizované směrování** (**centralized routing**). Jeho výhodou je možnost optimálního rozhodování na základě znalosti skutečného stavu celé sítě. Problém je ovšem v tom, že má-li být centralizované směrování opravdu adaptivní, tedy má-li průběžně reagovat na aktuální stav sítě, musí být vyhledávání nejvhodnějších cest prováděno dostatečně často. Vlastní hledání cest je samo o sobě operací značně náročnou na výpočetní kapacitu, a má-li se často opakovat, dokáže plně zaměstnat i velmi výkonný počítač. Jsou zde však ještě i další problémy - co například dělat v případě výpadku směrovacího centra? Nezanedbatelná není ani zátěž přenosových cest, kterou představuje neustálý přísun aktuálních informací o stavu sítě do směrovacího centra, stejně tak jako zpětná distribuce výsledků.

Izolované směrování

Alternativou k centralizovanému směrování je tzv. **izolované směrování** (**isolated routing**), založené na myšlence, že rozhodovat o nejvhodnější cestě si bude každý uzel sám za sebe, a to na základě takových informací, které dokáže získat sám, bez

spolupráce s ostatními uzly. Jednou z možností realizace je algoritmus, nazvaný příznačně **algoritmem horké brambory** (**hot potato algorithm**). Jak jeho název dává tušit, snaží se uzel zbavit každého paketu resp. datagramu co možná nejrychleji. Sleduje proto počet paketů, které čekají ve frontě na odeslání jednotlivými směry, a nový paket zařadí do té fronty, která je momentálně nejkratší. Uzel se tedy nerozhoduje podle adresy, ani nehledá nejkratší cestu pro přenos paketu, pouze se jej snaží co nejrychleji zbavit ve víře, že po jisté době přeci jen dojde ke svému cíli. V praxi se ovšem algoritmus horké brambory spíše kombinuje s jinými algoritmy resp. metodami - nejčastěji je používán jako jejich doplněk, který se uplatní až v okamžiku, kdy počet paketů v některé frontě překročí určitou únosnou mez.

Zpětné učení

Jiným příkladem izolovaného směrování je tzv. **metoda zpětného učení** (**backward learning**). Předpokládá, že každý uzel si do svých směrovacích tabulek průběžně poznamenává, ze kterého směru dostává pakety, pocházející od jiných uzlů. Tím se postupně "učí", ve kterém směru se tyto uzly nalézají. Když pak sám potřebuje odeslat nějaký paket jinému uzlu, vyšle jej tím směrem, ze kterého dříve přijal paket, pocházející od téhož uzlu.

Problémem je ovšem vrozený optimismus metody zpětného učení. Dojde-li k výpadku určité přenosové cesty, kterou se jednotlivé uzly již "naučily", vůbec ji nezaznamenají. Prakticky jediným možným řešením je pak pravidelné "zapomínání".

Záplavové směrování

Extrémní formou izolovaného směrování je tzv. **záplavové směrování** (**flooding**). Předpokládá, že přijatý paket je znovu odeslán všemi směry kromě toho, odkud sám přišel.

Zřejmou výhodou je maximální robustnost, díky které se záplavové směrování dokáže vyrovnat prakticky s jakýmkoli výpadkem. Zaručuje také, že každý paket je vždy doručen tou nejkratší možnou cestou. Nevýhodou je ale vznik velkého množství duplicitních paketů, které výrazně zvyšují zátěž existujících přenosových cest, a které je třeba následně rušit.

V praxi se proto používá spíše tzv. **selektivní záplavové směrování** (**selective flooding**), při kterém není každý paket znovu vysílán všemi směry, ale pouze těmi, které jsou alespoň přibližně orientovány ke konečnému příjemci paketu.

Distribuované směrování

Metody izolovaného směrování staví na předpokladu, že jednotlivé uzly nebudou zatěžovat přenosové cesty vzájemnou výměnou informací o stavu sítě. To je ale někdy zbytečně přísným omezením.

Pokud jej odstraníme, dostaneme tzv. **distribuované směrování** (**distributed routing**). To předpokládá, že jednotlivé uzly si pravidelně vyměňují informace o stavu sítě, a podle nich si pak samy volí příslušné cesty.

Jakmile umožníme výměnu stavových informací mezi jednotlivými uzly sítě, můžeme vcelku efektivně implementovat distribuovanou verzi algoritmu hledání nejkratších cest v síti, který bychom zřejmě používali v případě centralizovaného směrování a jediného směrovacího centra. Naznačme si nyní myšlenku tohoto distribuovaného algoritmu.

Nejprve si však musíme ujasnit, co vlastně bude pro nás měřítkem "délky" nějaké cesty. Může to být například počet meziuzlů, kterými cesta prochází. Tím však dáváme každému existujícímu spoji mezi dvěma uzly stejnou jednotkovou váhu resp. délku. Realističtější je přiřadit vhodné ohodnocení (délku) každému přímému spoji mezi dvěma uzly, a délku výsledné cesty pak chápat jako součet délek jejích jednotlivých částí. Délka spoje přitom může odrážet jeho přenosovou rychlost, cenu za jednotku přenesených dat, zpoždění při přenosu, délku výstupních front apod.

Představme si nyní síť dle obrázku 34.1. a/, včetně "délek" jednotlivých přímých spojů. Každý uzel předem zná svou "vzdálenost" od všech svých sousedů, a tak si ji ve své směrovací tabulce vyznačí. Svou vzdálenost (délku nejkratší cesty) od ostatních uzlů však ještě nezná, a tak ji zatím považuje za nekonečnou (na obrázku naznačeno vyšrafováním). Počáteční stav tabulek ukazuje obrázek 34.1. b/.

Vlastní algoritmus distribuovaného výpočtu pak probíhá v opakujících se krocích. V každém z nich se každý uzel dotáže svých bezprostředních sousedů, jaké jsou jejich vzdálenosti od ostatních uzlů, a podle toho si pak odvozuje i své vlastní vzdálenosti od těchto uzlů.

Uvažme příklad uzlu E. Ten se v prvním kroku od svého souseda C dozví, že jeho vzdálenost od uzlu B je 2. K ní si uzel E připočte svou vzdálenost od uzlu C, tj. 1, a do své směrovací tabulky si poznačí, že cesta do uzlu B vede přes uzel C a má délku 3. Zatím však nezná všechny možné cesty do uzlu B, a tak neví, zda je to cesta nejkratší. Proto se bude v dalších krocích znovu ptát všech svých sousedů, zde přes ně nevede cesta ještě kratší.

Obecně si každý uzel vždy volí minimum z toho, co "již umí" on sám (tj. z cesty, kterou již má poznačenu ve své směrovací tabulce), a co "umí" jeho sousedé (samozřejmě s uvážením své vzdáleností od těchto sousedů). Příkladem může být opět uzel E, který si již na počátku do své směrovací tabulky zanese, že jeho vzdálenost od uzlu D je 5 (což je délka jejich přímého spojení, viz obr. 34.1 a/). Již v prvním kroku však od svého souseda C zjistí, že jeho vzdálenost od uzlu D je jen 2. Když si k tomu připočítá svou vzdálenost od uzlu C (tj. 1), vyjde mu, že cesta do uzlu D, vedená přes uzel C, je kratší. Tuto skutečnost si pak poznačí do své směrovací tabulky (viz obrázek 43.1 b/ a c/, tabulka uzlu E, položka D).

.PI OBR34_1.TIF, 20, 40, 10 .PI OBR34_2.TIF, 20, 40, 10

Obr. 34.1.: Představa distribuovaného výpočtu směrovacích tabulek a/ topologie sítě a délky spojů

b/ počáteční stav směrovacích tabulek

c/ stav směrovacích tabulek po prvním kroku

d/ "ustálený" stav směrovacích tabulek

Na obrázku 34.1. d/ je pak stav tabulek po několika krocích distribuovaného algoritmu, kdy již nedochází k žádným změnám ve směrovacích tabulkách. Zde by algoritmus mohl končit, v praxi se však budou jeho kroky neustále opakovat, aby obsah směrovacích tabulek mohl reagovat na průběžné změny v síti.

Právě naznačený algoritmus distribuovaného směrování byl používán v síti ARPA (základu dnešní sítě Internet), a jednotlivé kroky algoritmu zde probíhaly s intervalem 640 milisekund. Ukázalo se však, že režie je přeci jen příliš vysoká - že vzájemné předávání informací mezi sousedními uzly (které vlastně představuje předávání celých směrovacích tabulek) neúnosně zatěžovalo dostupné přenosové cesty na úkor "užitečných" dat. Proto byl uvedený algoritmus distribuovaného směrování v síti ARPA nahrazen jiným - o něm si ale budeme povídat později.

35/ Transportní vrstva

Hlavním úkolem transportní vrstvy referenčního modelu ISO/OSI je poskytovat efektivní přenosové služby své bezprostředně vyšší (tj. relační) vrstvě. Tyto služby přitom mohou mít spojovaný (connection-oriented) i nespojovaný (connectionless) charakter. Stejný charakter a stejnou podstatu však mají i služby síťové vrstvy, které transportní vrstva sama využívá. Do značné míry analogické jsou v obou vrstvách i mechanismy adresování a řízení toku dat. Zákonitě se pak nabízí otázka, proč je vlastně nutná samostatná transportní vrstva, když alespoň na první pohled nenabízí nic principiálně jiného, než vrstva síťová?

Odpověď na tuto otázku je skryta v obrázku, který jsme si ukázali již ve 32. dílu, a který nyní použijeme znovu jako obrázek 35.1. Ukazuje totiž, že komunikace dvou

.PI OBR35_1.TIF, 20, 40, 10

Obr. 35.1.: Představa průchodu dat vrstvami

koncových účastníků se transportní vrstva zúčastňuje jen na obou koncových počítačích, zatímco v jednotlivých meziuzlech se na přenosu podílí jen tři nejnižší vrstvy - fyzická, linková a síťová. V případě rozlehlých sítí jsou tyto mezilehlé uzly (též: uzly IMP, viz 32. díl) často jen přepojovacími uzly v rámci komunikační podsítě, a vyšší vrstvy u nich nemusí být vůbec realizovány. Tak je tomu například u většiny veřejných datových sítí, které jsou provozovány nejrůznějšími spojovými organizacemi. A zde je právě kámen úrazu - je-li komunikační podsíť majetkem spojové organizace, musí uživatelé a jejich koncové (hostitelské) počítače vystačit s tím, co jim komunikační podsíť prostřednictvím své síťové vrstvy nabízí. Chtějí-li něco jiného - například spolehlivou přenosovou službu místo nespolehlivé, spolehlivější místo méně spolehlivé, spojovanou místo nespojované apod. - musí si vše potřebné zajistit sami. A to právě v transportní vrstvě.

K čemu slouží transportní vrstva?

Transportní vrstva je tedy v referenčním modelu ISO/OSI především proto, aby vyšším vrstvám poskytovala kvalitnější přenosové služby, než jaké ve skutečnosti dokáže poskytovat vrstva síťová. Současně s tím pak může uživatele resp. vyšší vrstvy odstiňovat od konkrétních specifik používané komunikační podsítě, od její přenosové technologie, a především pak od všech nedokonalostí podsítě. Transportní vrstva je tedy vlastně rozhraním mezi poskytovateli přenosových služeb (komunikační podsítí) a jejich uživateli (nejvyššími třemi vrstvami). Je také "poslední instancí", která může změnit nespojovaný charakter poskytované přenosové služby na spojovaný a naopak. To je možné udělat také na úrovni síťové vrstvy (jak jsme si naznačili ve 33. dílu

našeho seriálu), nikoli však již na úrovni vyšších vrstev, které jsou uživateli těchto služeb (viz též obrázek 35.2.)

.PI OBR35_2.TIF, 20, 40, 10

Obr. 35.2.: Spojované a nespojované služby v ISO/OSI modelu

Síťové vs. transportní spojení

K lepšímu pochopení úlohy transportní vrstvy nám pomůže bližší přiblížení rozdílu mezi transportním a síťovým spojením. **Transportní spojení** (**transport connection**) je spojení, které vzniká mezi dvěma entitami transportní vrstvy (v případě spojovaných služeb). Ve skutečnosti je však realizováno prostřednictvím nižších vrstev, tedy prostřednictvím **síťových spojení** (**network connections**) jako spojení mezi dvěma entitami na úrovni síťové vrstvy. Standardně se každé jednotlivé transportní spojení realizuje prostřednictvím jednoho síťového spojení. Pokud ale transportní spojení požaduje vyšší přenosovou kapacitu, než jakou dokáže zajistit jedno síťové spojení, může být jedno transportní spojení zajišťováno pomocí více spojení síťových. Transportní vrstva se pak stará o rozdělení přenášených dat mezi jednotlivá síťová spojení tak, aby to pro vyšší vrstvy bylo transparentní (tj. neviditelné).

Může však nastat i opačný případ. Pokud je zapotřebí vytvořit více transportních spojení s relativně malými nároky na přenosovou kapacitu, může být vždy několik z nich realizováno prostřednictvím jediného síťového spojení. Takováto potřeba nastává např. při použití více terminálů, z nichž každý vyžaduje samostatné transportní spojení se vzdálených počítačem. Jsou-li ale tyto terminály používány spíše příležitostně a jsou-li fyzicky blízko sebe, může být dosti neekonomické zřizovat pro každý z nich samostatné síťové spojení. Potřebné přepínání (multiplexování) jednoho sdíleného síťového spojení mezi více transportních spojení pak zajišťuje opět transportní vrstva.

Kvalita služeb síťové vrstvy

Pro transportní vrstvu je velmi podstatné, jakou kvalitu může předpokládat u služeb, které jí poskytuje síťová vrstva. Této kvalitě pak musí být přizpůsobeny přenosové protokoly transportní vrstvy, mají-li vyšší vrstvy odstiňovat od všech specifik a nedokonalostí komunikační podsítě resp. síťové vrstvy.

Služby síťové vrstvy se v této souvislosti rozdělují na tři kategorie - viz tabulka 35.3. Do kategorie A patří síťové služby, které ztrácí či přenáší s chybou jen zcela zanedbatelné procento paketů, a u kterých prakticky nedochází k výpadkům již jednou navázaných spojení. Takovéto síťové služby lze považovat za (téměř) dokonalé, a transportní vrstva pak má při jejich použití nejméně práce. Kvality kategorie A dosahují síťové služby některých lokálních sítí, zatímco v případě rozlehlých sítí je kategorie A vzácnější než šafrán.

Kategorie Vlastnosti

A bez chyb při přenosu paketů a

	bez výpadků spojení
B	bez chyb při přenosu paketů, s výpadky spojení
C	s chybami při přenosu paketů, s výpadky spojení

Tabulka 35.3. Kategorie přenosových služeb na úrovni síťové vrstvy

Do kategorie B spadají síťové služby, které jsou z hlediska přenosu jednotlivých paketů stejně spolehlivé jako kategorie A, ale u kterých dochází častěji k výpadkům spojení. Tedy k situacím, kdy z různých důvodů (přetížení, poruchy hardwaru, chyby v softwaru apod.) dochází k předčasnému ukončení (výpadku) dříve navázaného spojení. Transportní vrstva se s tím dokáže účinně vyrovnat (opětovným navázáním nového síťového spojení) a existenci výpadků před vyššími vrstvami skrýt. Má s tím však samozřejmě "více práce", než v případě síťových služeb kategorie A. Do kategorie B spadá obvykle většina veřejných datových sítí na bázi doporučení X.25.

Do kategorie C pak patří všechny ostatní síťové služby. Transportní vrstva je musí považovat za nespolehlivé, a potřebnou spolehlivost zajistit sama. Protokoly transportní vrstvy, které musí využívat služby této kategorie, jsou pak samozřejmě nesložitější a nejkomplikovanější.

Třídy transportních protokolů

Referenční model ISO/OSI se vyrovnává s odlišnou kvalitou služeb na úrovni síťové vrstvy zavedení pěti různých tříd přenosových protokolů na úrovni transportní vrstvy (transportních protokolů) - viz tabulka 35.4.

Do třídy 0 (TP0, Transport Protocol class 0) patří transportní protokoly, které využívají síťové služby kategorie A. Jsou jednoduchou "obálkou" nad příslušnými síťovými protokoly, a navíc zajišťují prakticky jen nezbytné navazování a rušení transportních spojení.

Třída 1 (TP1) předpokládá použití síťových služeb kategorie B, a musí se tedy umět vyrovnat s případnými výpadky spojení na úrovni síťové vrstvy.

Třída 2 (TP2) předpokládá opět použití síťových služeb kategorie A, tedy takových, které může považovat za dostatečně spolehlivé - stejně jako v případě třídy 0. Transportní protokoly třídy 2 jsou však navíc schopné zajistit potřebné multiplexování jednoho síťového spojení mezi více spojení transportních (viz výše).

Třída 3 (TP3) je kombinací tříd 1 a 2. Předpokládá síťové služby kategorie B, a dokáže je multiplexovat mezi více spojení transportních.

Třída 4 (TP4) je určena pro použití nad síťovými službami kategorie C. Transportní protokoly této třídy musí

"počítat s nejhorším", a jsou tedy ze všech transportních protokolů nejsložitější a nejkomplikovanější.

Třída transportních protokolů	Kategorie síťových služeb	Funkce
TP0	A	jednoduchá obálka síťových protokolů
TP1	B	zotavení z výpadků síťových spojení
TP2	A	multiplexování jednoho síťového spojení mezi více transportních spojení
TP3	B	multiplexování a zotavení z výpadků
TP4	C	detekce a oprava chyb, zotavení z výpadků

Tabulka 35.4.: Třídy transportních protokolů

36/ Relační vrstva

Jak jsme si již naznačili dříve, čtyři nejnižší vrstvy referenčního modelu ISO/OSI jsou zaměřeny především na vlastní přenos dat mezi jednotlivými uzlovými počítači. Každá z nich přitom poskytuje "o něco více" než vrstva bezprostředně nižší - čtvrtá (tj. transportní) vrstva pak nabízí prostředky pro takovou komunikaci dvou koncových počítačů resp. účastníků, která je nezávislá na konkrétní komunikační podsíti, na její topologii i na dalších specifikách. Vyšší vrstvy referenčního modelu ISO/OSI jsou pak již orientovány spíše na potřeby síťových aplikací. Samy využívají přenosové služby transportní vrstvy, a přidávají k nim takové funkce a schopnosti, které by měly být užitečné pro většinu aplikací. Takový je tedy smysl existence páté (relační), šesté (prezentační), a vlastně i sedmé (aplikační) vrstvy ISO/OSI modelu.

Referenční model ISO/OSI tedy vychází z představy, že jednotlivé aplikace by měly mít k dispozici a měly by využívat ještě dokonalejší a obecnější služby, než jaké nabízí transportní vrstva. Jejich realizaci pak referenční model svěřuje relační (session) a prezentační (presentation) vrstvě. Mlčky tím ovšem předpokládá, že je

budou využívat všechny aplikace, nebo alespoň většina z nich - jen pak totiž má rozumný smysl realizovat je pro všechny aplikace společně. Existuje ale velké množství aplikací, které tento předpoklad nesplňují, a které plně vystačí s takovými službami, jaké nabízí již transportní vrstva. V tomto ohledu není jistě bez zajímavosti, že "konkurenční" soustava protokolů TCP/IP vychází z přesně opačné úvahy než referenční model ISO/OSI - svým aplikacím nabízí pouze přenosové služby na úrovni transportní vrstvy. Potřebuje-li některá aplikace resp. služba aplikační vrstvy služby obecnějšího charakteru, musí si je realizovat sama vlastními silami. To je také důvod, proč soustava protokolů TCP/IP, na které je vybudována např. síť ARPANET resp. dnešní Internet, s existencí relační a prezentační vrstvy vůbec nepočítá.

Relační vrstvu s takovými funkcemi, jaké jí přisuzuje referenční model ISO/OSI, neměla v době jeho zavedení žádná síť - snad s výjimkou sítě SNA firmy IBM, která má obdobné funkce realizovány ve více různých vrstvách.

Při vytváření samotného referenčního modelu ISO/OSI byla kolem relační vrstvy a jejích úkolů velká diskuse. Některé alternativní návrhy, které se v té době objevily, s relační vrstvou dokonce vůbec nepočítaly. Nakonec se však relační vrstva do referenčního modelu přeci jen prosadila. Spolu se svou bezprostředně nadřazenou (tj. prezentační vrstvou) však byla a nadále je relativně nejméně propracovanou vrstvou.

.cp9

Co je vlastně relace?

Pro správné pochopení smyslu relační vrstvy je dobré si nejprve ozřejmit onen poněkud vágní termín **relace** (**session**). Nejnázornější bude zřejmě analogie s telefonním hovorem - ten je třeba nejprve vytočit (čímž vzniká analogie transportního spojení), a pak je možné jeho prostřednictvím vést rozhovor (relaci) dvou účastníků.

Relaci tedy můžeme považovat za spojení mezi dvěma koncovými účastníky na úrovni bezprostředně vyšší, než je vrstva transportní. Obvykle je každé takovéto spojení (relace) zajišťováno prostřednictvím jednoho transportního spojení (tj. spojení na úrovni transportní vrstvy), které je zřizováno a rušeno při zřizování resp. rušení relace - viz obr. 36.1. a/. Je ovšem možný i takový případ, kdy jedno transportní spojení zajišťuje dvě nebo více po sobě jdoucích relací, viz obr. 36.1. b/. Použijeme-li opět analogii s telefonním hovorem, odpovídá tato situace tomu, že dva účastníci telefonního hovoru svůj rozhovor dokončí, ale místo zavěšení předají telefon jiné dvojici, která se může začít bavit o něčem zcela jiném - tedy zahájit nový rozhovor (relaci).

.PI OBR36_1.TIF, 20, 40, 10

Obr. 36.1.: Vztah relace a transportního spojení

a/ jedna relace - jedno transportní spojení

b/ více relací - jedno transportní spojení

c/ jedna relace - více transportních spojení

Podobně, jako se transportní vrstva snaží zakrýt případné výpadky spojení na úrovni síťové vrstvy, měla by se i relační vrstva dokázat vyrovnat s případným výpadkem transportního spojení, a zajistit pokračování relace prostřednictvím nově zřízeného

transportního spojení - viz obr. 36.1. c/. V naší "telefonní" analogii to odpovídá situaci, kdy v průběhu rozhovoru dvou účastníků je jejich hovor přerušen, a oni si musí zavolat znovu, aby svůj rozhovor dokončili.

Zajímavou odlišností relace od transportního spojení je i způsob jejího ukončení. V případě transportního spojení nabízí referenční model ISO/OSI jen prostředky pro jednostranné direktivní ukončení spojení, kterému druhá strana nemá možnost zabránit. Na úrovni relační vrstvy se však předpokládá spíše ukončení na základě vzájemné dohody obou zúčastněných stran. Přesněji na návrh jedné strany, který ale druhá strana má možnost odmítnou a zajistit si tak pokračování relace.

Řízení dialogu

Jedním z hlavních úkolů relační vrstvy je řízení dialogu mezi oběma koncovými účastníky. Tak jako při telefonním rozhovoru není možné (či alespoň slušné), aby oba účastníci mluvili současně, existují i v počítačových sítích takové aplikace, které vyžadují koordinované střídání obou zúčastněných při vysílání (například různé aplikace transakčního charakteru apod.).

Relační vrstva tento požadavek zajišťuje obvykle pomocí mechanismu předávání **pověření** k přenosu dat (**data token**) - vysílat data smí vždy jen ten, kdo vlastní tento pomyslný štafetový kolík (pověření resp. token), viz obrázek 36.2. b/. Relační vrstva přitom nabízí prostředky, pomocí kterých lze pověření předat, nebo si jej naopak vyžádat.

Obecně se na úrovni relační vrstvy rozlišují tři způsoby vedení dialogu - plně duplexní (v terminologii RM ISO/OSI: TWS resp. Two-Way-Simultaneous), poloduplexní (TWA resp. Two-Way-Alternate) a simplexní (One-Way). Mechanismus předávání pověření se přitom týká samozřejmě jen poloduplexního dialogu. Zdůrazněme si však, že způsob vedení dialogu nemá nic společného s duplexním, poloduplexním či ev. simplexním charakterem přenosového kanálu. Všechna spojení v referenčním modelu ISO/OSI jsou obecně plně duplexní, a umožňují tedy současný přenos dat oběma směry. Různé aplikace však z různých důvodů nemusí této možnosti využívat, a po plně duplexním spoji mohou vést jen poloduplexní dialog.

.cp9

.PI OBR36_2.TIF, 20, 40, 10

Obr. 36.2.: Představa vzájemné komunikace v rámci relace

a/ plně duplexní dialog

b/ poloduplexní dialog

Synchronizace

Dalším důležitým úkolem relační vrstvy je tzv. **synchronizace** (**synchronization**, též: **checkpointing**). K jejímu pochopení si nejprve představme následující situaci: příjemcem dat v rámci určité relace nechť je počítač, který si přijímaná data průběžně zapisuje na disk, případně je ihned tiskne na tiskárně. Dojde-li náhle k poruše disku či tiskárny (nebo jí např. dojde papír, toner apod.), může příjemce přijít o určitý objem

dat, které jinak v pořádku přijal (tj. které byly transportní vrstvou bezchybně doručeny).

Co je pak zapotřebí, je možnost "vrátit se o kousek zpět", a ztracená data přenést znovu. Tedy znovu dosáhnout potřebné synchronizace (vzájemného souladu). Relační vrstva řeší tento problém možností vkládat do přenášených dat kontrolní body (**synchronization points, checkpoints**). Příjemci pak umožňuje, aby si na vysílajícím vyžádal návrat k zadanému kontrolnímu bodu, a nové vyslání těch dat, která leží za tímto bodem.

Není ovšem úkolem relační vrstvy pamatovat si přenášená data tak, aby byl možný návrat k existujícím kontrolním bodům. Tato data si musí "pamatovat" sám jejich odesilatel, který služeb relační vrstvy využívá. Aby si ale nemusel uchovávat zbytečně velké objemy dat, rozlišuje relační vrstva dva druhy kontrolních bodů - hlavní (**major**) a vedlejší (**minor**). Rozdíl mezi nimi je ten, že přes vedlejší kontrolní bod se vracet lze, zatímco přes hlavní kontrolní bod již nikoli. Pro vysílajícího z toho pak vyplývá, že si musí "pamatovat" přenášená data jen od posledního hlavního kontrolního bodu - viz obr. 36.3.

.PI OBR36_3.TIF, 20, 40, 10

Obr. 36.3.: Představa kontrolních bodů

37/ Prezentační vrstva

Pět nejnižších vrstev referenčního modelu ISO/OSI dělá vše pro to, aby přenášená data vždy dorazila ke svému koncovému příjemci přesně v takové podobě, v jaké byla vyslána. Stejná "podoba" však ještě nezaručuje, že pro příjemce nebudou jedna a tatáž data představovat něco jiného, než pro jejich odesilatele.

Obavy z různé interpretace přenášených dat nejsou zdaleka bezdůvodné. Stačí si uvědomit, že např. střediskové počítače firmy IBM používají pro kódování znaků kód EBCDIC, zatímco drtivá většina ostatních používá ke stejnému účelu kód ASCII. Ke znázornění celých čísel se znaménkem používá většina počítačů tzv. dvojkový doplňkový kód, ale např. počítače CDC Cyber pracují s tzv. jedničkovým doplňkovým kódem. Mikroprocesory 80x86 firmy Intel číslují jednotlivé byty (ve slovech, dvojslovech atd.) jedním směrem, zatímco například mikroprocesory řady M68000 firmy Motorola číslují jednotlivé byty přesně opačně. Velmi časté jsou pak také odlišnosti například ve formátu čísel v pohyblivé řádové čárce, odlišné rozsahy zobrazitelných celých čísel (dané počtem k tomu vyhrazených bitů) apod.

Různé počítače tedy v obecném případě používají různé způsoby vnitřní reprezentace dat. Mají-li si takové počítače svá data korektním způsobem vzájemně předávat, musí být vhodným způsobem zajištěny jejich nezbytné konverze. A ty má v referenčním modelu ISO/OSI na starosti právě **prezentační vrstva** (**presentation layer**).

Prezentační vrstva se tedy stará o to, aby například celé číslo bez znaménka s hodnotou 234 bylo přijato opět jako celé číslo bez znaménka s hodnotou 234, a ne např. jako celé číslo se znaménkem s hodnotou -22. Není však již úkolem prezentační vrstvy zabývat se tím, co toto číslo znamená. Zda jde např. o počet osob, o číslo

stránky v knize, o procentuální výši inflace či něco docela jiného. To přísluší až vlastním aplikacím, které jsou zdrojem resp. příjemcem těchto dat.

Pro zajištění nezbytných konverzí na úrovni prezentační vrstvy se nabízí dvě základní možnosti, které naznačuje obrázek 37.1. První z nich představuje vzájemné přímé přizpůsobení stylu "každý s každým", při kterém jsou přenášená data konvertována jen jednou - jsou-li ovšem k dispozici nezbytné konverzní rutiny pro libovolnou dvojici komunikujících uzlů. Ve druhém případě jsou přenášená data konvertována dvakrát: ze tvaru, se kterým pracuje odesilatel, jsou nejprve převedena do společného "mezitvaru", a z něj pak do takového tvaru, s jakým pracuje jejich příjemce. Nevýhoda dvojí konverze je zde kompenzována tím, že pro každou konkrétní reprezentaci dat, se kterou pracuje nějaký počítač, stačí jediná konverzní utilita pro jejich převod z/do společného "mezitvaru".

.PI OBR37_1.TIF, 20, 40, 10

Obr. 37.1.: Možnosti konverze

a/ typu "každý s každým"

b/ se společným "mezitvarem"

Referenční model ISO/OSI předpokládá právě tuto druhou variantu se společným mezitvarem. Podívejme se proto na její podstatu poněkud podrobněji.

Chtějí-li vzájemně spolupracovat dvě různé síťové aplikace, musí se nejprve domluvit na společných datových strukturách, které budou používat - tedy například na tom, že datum budou reprezentovat jako záznam (record) tvořený třemi položkami (DEN, MESIC a ROK), které jsou samy o sobě celými čísly bez znaménka. Tyto datové struktury je ovšem nutné vyjádřit tak, aby jejich popis byl pro obě strany srozumitelný, a obě strany si jej také stejně vykládaly. Kdyby byly všechny síťové aplikace psány v jediném vyšším programovacím jazyku, stačilo by použít právě tento jazyk. Předpoklad použití jediného programovacího jazyka však nebyl, není a zřejmě nikdy nebude v praxi splněn, a tak bylo nutné vytvořit pro potřeby formálního popisu dat a datových struktur zvláštní jazyk, který byl nazván **ASN.1** (**Abstract Syntax Notation**). Umožňuje definovat jednotlivé datové položky, stanovit jejich typ (tj. určit, zda jde např. o celé číslo se znaménkem, znakový řetězec či logickou hodnotu apod.), přidělit jim jméno (identifikátor), a také sestavit z jednoduchých datových položek obecnější datové struktury typu záznam, pole, seznam, množina apod.

.cp9

Jazyk ASN.1, který vzdáleně připomíná jazyk Pascal, je tedy jazykem pro formální popis dat - což se v terminologii ISO/OSI modelu označuje jako **abstraktní syntaxe** (**abstract syntax**). Abstraktní proto, že ještě neurčuje žádný konkrétní způsob reprezentace těchto dat. Pro potřeby vlastního přenosu je ale samozřejmě nutné veškerá data vhodným způsobem zakódovat. Způsob kódování datových struktur jazyka ASN.1 pro potřeby jejich přenosu pak určuje samostatná norma organizace ISO (IS 8825). Formát skutečně přenášených dat se přitom v terminologii ISO/OSI modelu označuje jako **přenosová syntaxe** (**transfer syntax**). Její konkrétní tvar je založen na myšlence, že každá jednotlivá položka dat by měla být samoidentifikující, tedy měla by sebou nést i informaci o vlastním typu. Každá datová položka proto má

při přenosu tři části, které po řadě určují její typ, délku a vlastní obsah (viz obrázek 37.2.).

.PI OBR37_2.TIF, 20, 40, 10

Obr. 37.2.: Představa fungování prezentační vrstvy

Způsob fungování prezentační vrstvy názorně ilustruje obrázek 37.2. Kdykoli chce nějaká entita aplikační vrstvy zaslat určitá data své partnerské entitě na jiném uzlovém počítači, předá "své" prezentační vrstvě jednak vlastní data, která si přeje odeslat, a jednak jejich popis v jazyce ASN.1 (který definuje jejich abstraktní syntaxi). Prezentační vrstva na základě tohoto popisu dokáže správně interpretovat jednotlivé položky dat (určit mj. jejich typ a velikost), a na základě toho je pak zakódovat do takového tvaru, který je vhodný pro přenos, a který si sebou nese potřebné informace o typu a formátu přenášených dat (tj. převést je do přenosové syntaxe). Prezentační vrstva na straně příjemce pak díky tomu dokáže správně určit typ a formát přijatých dat, a v případě potřeby provést nezbytné konverze. Jestliže například přenosová syntaxe počítá s vyjádřením celých čísel se znaménkem ve dvojkovém doplňku, ale příjemce používá ke stejnému účelu jednotkový doplněk, může prezentační vrstva příjemce provést nezbytné konverze ještě dříve, než přijatá data předá své bezprostředně vyšší (tj. aplikační vrstvě).

Prezentační vrstvy příjemce a odesilatele se však nejprve musí shodnout na tom, jaké datové struktury si vlastně budou předávat, a jakou budou pro ně používat přenosovou syntaxi. Proto se musí obě strany na začátku vzájemného spojení (přesněji: při zahajování relace) nejprve dohodnout na jednom nebo několika tzv. **kontextech**, jak se v terminologii ISO/OSI modelu nazývá přiřazení přenosové syntaxe k syntaxi abstraktní. V průběhu relace se pak mohou mezi těmi kontexty, na kterých se oba dohodli, dokonce přepínat.

.PI OBR37_3.TIF, 20, 40, 10

Obr. 37.3.: Představa překladače jazyka ASN.1

V současné době je praktické používání jazyka ASN.1 značně usnadněno existencí překladačů z tohoto jazyka do obvyklých vyšších programovacích jazyků, např. do Pascalu či jazyka C. Jak názorně ukazuje obrázek 37.3., je vstupem tohoto překladače popis datových struktur v jazyku ASN.1, a výstupem jednak ekvivalentní popis v příslušném cílovém jazyku (tedy např. v C či v Pascalu), který pak lze přilinkovat ke zdrojovému tvaru vlastní aplikace, a dále také kódovací a dekódovací rutiny (určené pro potřeby prezentační vrstvy), které převádí datové struktury z příslušného cílového jazyka přímo do jejich přenosové syntaxe resp. obráceně.

Hlavním úkolem prezentační vrstvy je tedy zajištění nezbytných konverzí přenášených dat. Není to ovšem úkol jediný - na úrovni prezentační vrstvy může být například řešeno také zabezpečení přenášených dat pomocí **šifrování** (**encryption**), které ovšem lze realizovat i na úrovni fyzické nebo transportní vrstvy. Pro

minimalizaci objemu přenášených dat pak může být na úrovni prezentační vrstvy zajišťována i jejich **komprimace** (**compression)**.

38/ Aplikační vrstva

Bylo by jistě nošením dříví do lesa tvrdit, že počítače jsou zde proto, aby dělaly něco užitečného, a vyjmenovávat, co všechno to může být. Přesto neuškodí přesněji vymezit pojem **aplikace** (aplikačního programu, aplikačního procesu, aplikační úlohy) jako té části programového vybavení, která zajišťuje ony "užitečné funkce", kvůli kterým se počítače vůbec používají. Protipólem je pak zbývající část programového vybavení, která zajišťuje fungování počítače jako takového, a jednotlivým aplikacím vytváří takové prostředí, ve kterém mohou úspěšně pracovat.

Jde-li o samostatný počítač, jsou aplikacemi jednotlivé uživatelské programy, a "zbývající" částí programového vybavení je operační systém, který uživatelským programům zprostředkovává využití nejrůznějších zdrojů a prostředků daného počítače - disků, operační paměti, klávesnice, displeje, dalších V/V zařízení apod. V případě uzlů počítačové sítě je pak odlišnost pouze v tom, že síťový operační systém, který v sobě implementuje jednotlivé vrstvy vrstvového síťového modelu, zprostředkovává aplikacím navíc ještě všechno to, co vlastní síť nabízí - tedy především možnost komunikace s aplikacemi, běžícími na jiných uzlových počítačích, případně i přístup k technickým prostředkům jiných uzlových počítačů atd.

Nyní již si můžeme přesněji vymezit funkci **aplikační vrstvy** (**application layer**) v rámci referenčního modelu ISO/OSI. V prvním přiblížení lze říci, že je určena k tomu, aby aplikacím poskytovala přístup do prostředí sítě na bázi ISO/OSI modelu, tedy aby fungovala jako jakési okno (či brána) mezi aplikacemi na různých uzlových počítačích, které si chtějí vzájemně vyměňovat nějaké informace.

Představy o podstatě a mechanismu fungování aplikační vrstvy se však v průběhu existence referenčního modelu ISO/OSI vyvíjely a značně zpřesňovaly. Původní představu, která se objevila v první verzi referenčního modelu, ukazuje obrázek 38.1. Zde se předpokládalo, že jednotlivé uživatelské aplikace budou zasahovat až do vlastní aplikační vrstvy - přesněji: ty části aplikací, které se bezprostředně týkají sítě a jejího využití, budou přímou součástí aplikační vrstvy (v terminologii ISO/OSI modelu se tyto části aplikací označovaly jako **aplikační entity**, resp. **application entities**). Zbývající části aplikací již nebyly považovány za součást "síťového prostředí ISO/OSI modelu".

Tato představa znamenala, že jednotlivé aplikace (resp. ty jejich části, které tvořily aplikační entity) si musely samy zajišťovat všechny služby, potřebné na úrovni aplikační vrstvy (a to pomocí služeb, poskytovaných resp. zprostředkovaných prezentační vrstvou). Referenční model ISO/OSI tyto služby na úrovni aplikační vrstvy nijak podrobněji nespecifikoval, pouze je vymezil jako "takové, které jsou potřeba pro vzájemnou komunikaci mezi otevřenými systémy, a nejsou zajišťovány na nižších úrovních". Nespecifikoval samozřejmě ani žádné protokoly, prostřednictvím kterých by služby aplikační vrstvy měly být realizovány.

.PI OBR38_1.TIF, 20, 40, 10

Obr. 38.1.: Původní představa aplikační vrstvy ISO/OSI

Jednotlivé protokoly pro aplikační vrstvu vznikaly až dodatečně, v průběhu práce na implementaci síťových aplikací různého typu. Přitom se ale ukázalo, že většina těchto aplikací má mnoho společného, a tudíž se nevyplatí, aby si každá z nich vždy znovu implementovala to, co na úrovni aplikační vrstvy potřebuje. Proto se původní představa referenčního modelu změnila: zajištění služeb na úrovni aplikační vrstvy bylo svěřeno aplikačním entitám, které ale již nejsou součástí jednotlivých aplikací, ale jsou naopak součástí síťového programového vybavení.

Diferenciace na úrovni aplikační vrstvy pak pokračovala ještě dále. Zjistilo se totiž, že takto koncipované aplikační entity je nejvýhodnější sestavovat z ještě menších stavebních bloků, které zajišťují jednotlivé dílčí funkce resp. služby. Tyto stavební bloky se v současné terminologii referenčního modelu ISO/OSI označují jako prvky **ASE** (**Application Service Elements**). Jsou v zásadě dvojího druhu: takové, které zajišťují služby, potřebné pro podporu aplikací různých typů (pak jde o tzv. prvky **CASE**, **Common Application Service Element**), a konečně takové, které realizují specifické služby, potřebné jen pro jeden konkrétní typ aplikací (označované jako prvky **SASE, Specific Application Service Element**) - viz obr. 38.2. a 38.3.

.PI OBR38_2.TIF, 20, 40, 10

Obr. 38.2.: Současná představa aplikační vrstvy ISO/OSI

.PI OBR38_3.TIF, 20, 40, 10

Obr. 38.3.: Prvky SASE a CASE

Vezměme si jako příklad otázku vzájemné komunikace dvou aplikací. Ta může být realizována buď jako spojovaná, prostřednictvím (logického) spojení mezi dvěma aplikačními entitami, nebo jako nespojovaná, zajišťovaná jednoduchou výměnnou zpráv. Logické spojení na úrovni aplikační vrstvy, v terminologii ISO/OSI modelu označované jako **asociace** (**association**), však musí někdo navázat, udržovat jej, a posléze zase zrušit. To má na starosti jeden z možných prvků ASE, konkrétně **ACSE** (**Association Control Service Element**), který patří mezi "podpůrné" prvky ASE, tj. mezi prvky CASE.

Nespojovaná komunikace na úrovni aplikační vrstvy, realizovaná jako výměna krátkých zpráv, má vesměs charakter **vzdáleného volání procedur** (**remote procedure call**, v terminologii ISO: **vzdálených operací, remote operation**), při kterém přenášené zprávy žádají o provedení určité akce (provedení resp. volání procedury), resp. vrací její výsledky. Pro zajištění takovéhoto způsobu komunikace je pak určen jiný druh podpůrného prvku (prvku CASE), a to **ROSE** (**Remote Operations Service Element**).

Dalším příkladem podpůrného prvku je prvek **CCR** (**Commitment, Concurrency and Recovery**), který je určen především pro koordinaci vzájemné součinnosti více uzlů a pro implementaci transakčního zpracování v sítích. Umožňuje dosáhnout toho, aby posloupnost operací, vyžádaná jedním uzlem a prováděná na jiném uzlu, se vždy provedla buďto celá, nebo se neprovedla vůbec, bez ohledu na případné výpadky, poruchy apod.

Mezi specifické aplikační služby, které mají na starosti prvky CASE, patří například přenos souborů, elektronická pošta, vzdálené terminálové relace apod. Referenční

model ISO/OSI přitom předpokládá, že tyto služby budou implementovány tak, aby se vůči vlastním aplikačním procesům "tvářily" vždy stejně, jako jediné **virtuální zařízení** (**virtual device**). Nejmarkantnější je tento přístup na způsobu, jakým se model ISO/OSI vyrovnává se značnou různorodostí používaných terminálů, lišících se svými parametry, způsobem ovládání apod. Mechanismus, umožňující terminálu jednoho uzlového počítače vystupovat v roli (lokálního) terminálu jiného počítače, počítá s existencí tzv. **virtuálního terminálu**. Jde vlastně o abstrakci, o hypotetický terminál, který nemusí ve skutečnosti vůbec existovat, ale který se vždy chová stejně. Uzlový počítač pak může předpokládat, že pracuje vždy jen s tímto jediným typem (virtuálního) terminálu, zatímco ve skutečnosti pracuje s terminálem jiným. Potřebné přizpůsobení mezi virtuálním terminálem a skutečně používaným terminálem pak zajišťuje až konkrétní aplikační proces, který "svůj" terminál dobře zná a dokáže ovládat.

Podobně je tomu i v případě přenosu souborů mezi odlišnými a vzájemně neslučitelnými souborovými systémy (které se mohou lišit například ve vnitřní organizaci souborů a adresářů, v konvencích pro jejich jména apod.). Zda se opět zavádí společný "mezitvar" v podobě virtuálního systému souborů, se kterým aplikační entity dokáží pracovat.

Pokud síťová aplikace potřebuje jiné služby, než jaké jsou jí nabízeny v souvislosti s představou virtuálního zařízení, musí si je sama podle svých konkrétních potřeb upravit. V rámci většiny aplikací lze proto ještě vymezit vrstvu, která zajišťuje potřebné přizpůsobení resp. zamapování virtuálního zařízení z/do skutečného - tedy např. "mapování" virtuálního terminálu do konkrétního reálného terminálu, "mapování" reálného systému souborů do virtuálního apod. V terminologii ISO/OSI modelu se tato vrstva označuje jako (uživatelský) prvek **UE** (**User Element**) - viz obrázek 38.2.

Pro specifické služby na úrovni aplikační vrstvy (poskytované prvky SASE, viz výše), byly a jsou postupně vytvářeny potřebné protokoly. Většina z nich dnes již má formu mezinárodních standardů resp. norem (IS, International Standards), jiné jsou zatím ještě ve stádiu návrhů norem (DIS, Draft International Standard). Jejich stručný přehled obsahuje tabulka 38.4., která uvádí také obdobně zaměřené aplikační protokoly v "konkurenční" soustavě protokolů TCP/IP.

zkratka	název protokol	je určen pro:	zaměřený protokol
TCP/IP			
VT	Virtual Terminal	virtuální terminály	Telnet
FTAM	File Transfer, Access and Management	přenos a sdílení souborů	FTP

MOTIS	Message Oriented Text Interchange Standard	elektronickou poštu[1]
SMTP		
CMIP	Common Management Information Protocol	správu sítí
SNMP		
JTM	Job Transfer and Manipulation	zadávání úloh na dálku,
MMS	Manufacturing Messaging Service	přenos zpráv v prostředí výroby
RDA	Remote Database Access	přístup do vzdálených databází

Tabulka 38.4.: Protokoly aplikační vsrtvy ISO/OSI

39/ Vzájemné propojování sítí - I.

Při našem povídání o referenčním modelu ISO/OSI jsme až doposud uvažovali pouze v dimenzích jediné počítačové sítě, a neuvažovali její možné vazby na sítě jiné. Uživatelé však dnes stále naléhavěji volají po vzájemném propojování existujících sítí, které by bylo co možná nejuniverzálnější, ale současně i co nejméně "viditelné", tak aby o něm museli vědět co nejméně. V ideálním případě by měla zcela zaniknout jakákoli specifika jednotlivých sítí, aby se uživatel mohl domnívat, že pracuje v jedné jediné, všezahrnující síti. Podívejme se proto nyní, jaké jsou principiální možnosti vzájemného propojování sítí. V dalších dílech si pak ukážeme, jak k této problematice přistupuje referenční model ISO/OSI i některé další síťové architektury..

Vzájemným propojením dvou či více sítí stejného či různého typu vzniká větší celek, pro který má angličtina velmi výstižné označení: **internetwork**, nebo jen **internet** (zatímco **Internet** - s velkým I - je jméno celosvětové počítačové sítě resp. konglomerátu sítí, který vznikl v USA ze zárodečné sítě ARPANET, viz např. CW 7/92, str. 12). Čeština však vhodný termín pro označování vzájemně propojených sítí teprve hledá - viz rubrika **Co (ne)najdete ve slovníku**).

Problematika vzájemného propojování sítí a jejich součinnosti (v angličtině: **internetworking**) je sama o sobě značně rozsáhlá, rychle se vyvíjí, a je dnes velmi populární. V současné době existuje řada různých koncepcí toho, jak počítačové sítě navzájem propojovat. Pro jejich pochopení je ale klíčovým momentem poznání toho, že vzájemné propojení je možné realizovat na různých úrovních vrstvového síťového modelu - od fyzické až po aplikační vrstvu.

[1] ISO verze standardu CCITT X.400 pro elektronickou poštu

Základní myšlenka vzájemného propojení je naprosto triviální - dvě nebo více sítí se propojí prostřednictvím k tomu určených zařízení, obecně označovaných jako **relay**, v terminologii ISO též: **Intermediary System** resp. **IS**, případně: **Internetworking Unit** resp. **IWU**.

Opakovače

Zmíněné propojující zařízení přitom může být pouhým zesilovačem elektrických signálů, které jsou v síti přenášeny. Tak je tomu například v lokálních sítích typu Ethernet, které umožňují dosáhnout celkové délky kabeláže až 2,5 kilometru, ale jednotlivé připojovací obvody (tzv. transceivery) jsou schopné generovat elektrické signály s dosahem jen asi 500 metrů. Pak je nutné sestavovat celé kabelové vedení ze segmentů (souvislých úseků kabelů) délky maximálně 500 metrů (v případě tzv. tlustého Ethernetu, viz 28. díl našeho seriálu, resp. 200 metrů v případě tzv. tenkého Ethernetu). Jednotlivé segmenty se pak musí spojovat pomocí zařízení, která se v tomto případě nazývají **opakovače** (**repeaters**), a fungují právě a pouze jako zesilovače elektrických signálů. Opakovače tedy pracují na úrovni fyzické vrstvy (viz obr. 39.1. a/), neboť vzhledem k zesilování elektrických signálů "vnímají" jen jednotlivé přenášené bity, ale nikoli již celé bloky (rámce) dat.

.PI OBR39_1.TIF, 20, 40, 10

Obr. 39.1.: Představa opakovače (a/) a mostu (b/)

Mosty

Nevýhodou opakovačů je skutečnost, že "propouští" veškerý provoz z jednoho segmentu do druhého, i když by to vůbec nebylo nutné. Výhodnější by jistě byl případ, kdyby propojovací zařízení dokázalo správně rozpoznat, která data mohou zůstat "uvnitř" příslušné části sítě (segmentu), a nezatěžovat jimi provoz v ostatních segmentech. K tomu je ale nutné, aby propojovací zařízení "vnímalo" celé přenášené rámce a znalo jejich formát natolik, aby dokázalo zjistit fyzickou adresu jejich odesilatele a příjemce (a podle toho je pak propustit do sousedního segmentu či nikoli). To ovšem znamená, že takovéto propojovací zařízení, které se pak nazývá **most** (**bridge**), musí pracovat na úrovni linkové vrstvy (resp. na úrovni podvrstvy řízení přístupu k médiu (vrstvy MAC) v případě lokálních sítí dle standardů IEEE 802), kde již lze fyzické adresy příjemce a odesilatele rozpoznat.

Rozdíl mezi opakovačem a mostem spočívá dále i v mechanismu jejich fungování. Zatímco opakovač nemá paměť a přenášená data resp. signály zpracovává průběžně (je pro ně vlastně "průchozí"), most již pracuje na principu "store and forward" (přijmi a předej dál). Most tedy z každé strany průběžně přijímá jednotlivé datové rámce, a podle adres v nich se rozhoduje, zda je předá na opačnou stranu či nikoli. Existuje přitom více konkrétních postupů a algoritmů, které mohou mosty v této souvislosti používat.

Jednou z nejjednodušších variant je ta, při které most průběžně vyhodnocuje odesilatele jednotlivých rámců, a podle toho, ze kterého směru příslušný rámec přijal, si pak sám odvozuje umístění jednotlivých uzlů. Záhy se tak sám dokáže "naučit" topologii sítě. V době, kdy ji ještě nezná, jednoduše předává všechny rámce do všech ostatních segmentů.

Tato metoda je velmi atraktivní proto, že nevyžaduje žádné konfigurování mostu (který se všechno potřebné naučí sám). Jednotlivé uzly v síti přitom nemusí o jeho existenci vůbec vědět - proto se také takovýto typ mostu označuje jako tzv. **transparentní most** (**transparent bridge**). Lze jej ovšem použít jen v takových sítích, které mají přísně stromovitou strukturu, kdy mezi každými dvěma uzly existuje vždy jen jedna jediná cesta. Pro obecnější topologie jsou pak nutné jiné, složitější algoritmy práce mostů.

Pokud jde o technické provedení, mohou být opakovače i mosty konstruovány tak, aby vzájemně propojovaly jen dva segmenty, nebo také více segmentů najednou, viz obr. 39.2.b/. Pak jde o tzv. **vícevstupové opakovače** (**multiport repeater**) resp. **vícevstupové mosty** (**multiport bridge**).

.PI OBR39_2.TIF, 20, 40, 10

Obr. 39.2.: Varianty mostů

a/ dvouvstupový most (bridge)

b/ vícevstupový most (multiport bridge)

c/ vzdálený most (remote bridge)

Mosty se vyrábí i ve variantě tzv. **vzdálených mostů** (**remote bridge**). Od standardní varianty mostů (označovaných pro odlišení také jako **místní mosty** resp. **local bridges**) se vzdálené mosty liší v tom, že jde vlastně o dvě relativně samostatné "poloviny" mostu, příznačně nazývané **půlmosty** (**halfbridge**), viz obr. 39.2. c/, které jsou mezi sebou vhodně propojeny - např. pevným telefonním okruhem, optickým kabelem apod. Umožňují propojit dva segmenty sítě, které nejsou fyzicky blízko sebe. Takto lze například propojit dva segmenty lokální sítě ve dvou objektech na opačných stranách města, přičemž výsledný efekt je takový, že oba segmenty tvoří jedinou "logickou" síť (z pohledu síťové vrstvy a všech vyšších vrstev je totiž existence místních i vzdálených mostů transparentní).

Některé mosty pak mohou mít i schopnost selektivního filtrování některých rámců v závislosti na jejich odesilateli či příjemci, denní době, intenzitě provozu apod. Pak jde o tzv. **routing bridges**, které správcům sítí umožňují regulovat přenosy mezi jednotlivými segmenty - umožňují například zakázat v době "špičky" přístup z jednoho segmentu do jiného, a při poklesu intenzity provozu jej pak zase následně povolit.

.PI OBR39_3.TIF, 20, 40, 10

Obr. 39.3.: Představa páteřní sítě

Mosty i opakovače se tedy používají pro spojování jednotlivých segmentů lokálních sítí. Opakovače jako jednodušší a lacinější (a současně i rychlejší) se pak volí spíše tam, kde intenzita provozu není velká. Použití mostů (obecně složitějších, dražších a také pomalejších než opakovače) je naopak vhodné tam, kde je potřeba vzájemně

propojit segmenty s větší intenzitou provozu tak, aby se navzájem nezatěžovaly více, než je skutečně nezbytně nutné.

Velmi častou a oblíbenou topologií lokálních sítí sběrnicového typu Ethernet je zapojení s tzv. **páteří** (**backbone**), což je segment, určený především pro vzájemné propojení ostatních segmentů - viz obr. 39.3. Výhodou této topologie je skutečnost, že spojení mezi kterýmikoli dvěma body sítě prochází vždy nejvýše přes dva mosty. Například v budově o více patrech je možné vést jednotlivé segmenty horizontálně po patrech, a pomocí mostů je připojit na vertikálně vedenou páteř, která prochází všemi patry.

40/ Vzájemné propojování sítí - II.

V minulém dílu našeho seriálu jsme si naznačili, že počítačové sítě je možné vzájemně propojovat na různých úrovních vrstvového síťového modelu. Podrobněji jsme se pak zabývali opakovači (repeaters) a mosty (bridges), které realizují propojení na úrovni fyzické resp. linkové vrstvy. Dnes budeme pokračovat možnostmi propojování na vyšších úrovních.

Vraťme se však ještě na chvilku k mostům. Jak jsme si již uvedli, používají se tyto v lokálních sítích pro spojování jednotlivých segmentů, pracují na úrovni linkové vrstvy (přesněji: na úrovni podvrstvy MAC), a při své činnosti vychází pouze z fyzických adres skutečného odesilatele a příjemce jednotlivých rámců. Vlastní datový obsah jednotlivých rámců přitom nijak neinterpretují ani nemění. Tím jsou pro ně neviditelné veškeré informace, které do obsahu vlastního rámce zakódovaly protokoly vyšších vrstev, od síťové počínaje. Je jim ovšem také jedno, které konkrétní protokoly to byly. Jinými slovy: mosty jsou zcela transparentní pro protokoly vyšších vrstev. Dokáží tedy spolupracovat s jakýmikoli síťovými (a vyššími) protokoly, a přenášet jejich pakety bez toho, že by je jakkoli transformovaly či měnily. Jednotlivé segmenty, které jsou vzájemně propojeny prostřednictvím mostů, tvoří z pohledu síťové vrstvy (i všech vyšších) jediný logický celek, který má také jedinou společnou (síťovou) adresu.

Směrovač a jeho funkce

Jakmile však budeme požadovat, aby si jednotlivé segmenty lokálních zachovaly relativní samostatnost (například vlastní síťovou adresu, možnost samostatné správy apod.), nebo když potřebujeme vzájemně propojit lokální sítě různých typů, spojujeme-li dvě lokální sítě přes síť rozlehlou nebo vytváříme-li vzájemné propojení sítí se složitější topologií, musíme k tomu použít obecnější řešení, než jaké nabízí mosty. Potřebujeme propojovací zařízení, které již pracuje na úrovni síťové vrstvy, a nazývá se **směrovač** (**router**), viz obr. 40.1. a/. Teprve takovéto zařízení totiž "vnímá" vlastní obsah jednotlivých rámců (na úrovni linkové vrstvy), dokáže správně rozpoznat formát jednotlivých paketů, které jsou v rámcích přenášeny, a využít informace, které jsou v nich obsaženy.

Hlavní úkol směrovačů je vlastně shodný s úkolem síťové vrstvy - tedy postarat se o doručení paketů od jejich původního odesilatele až ke konečnému příjemci (viz 34. díl našeho seriálu). Směrovače tedy musí přijímat rozhodnutí o tom, kudy mají dále odeslat každý jednotlivý paket tak, aby se dostal ke svému cíli - tedy zajišťovat to, čemu se běžně říká **směrování** (**routing**). Musí nutně používat nějaký algoritmus směrování, na základě kterého svá rozhodnutí přijímají. Jak jsme si již také uvedli ve

34. dílu, může mít tento algoritmus a z něho vycházející směrování statickou povahu (tj. být nezávislé na okamžitém stavu sítě), nebo může mít naopak dynamickou povahu (a reagovat na průběžnou situaci). V tomto druhém případě, který je dnes nejčastější, pak ještě potřebuje vhodnou metodu resp. protokol, prostřednictvím kterého získává potřebné informace o stavu sítě.

.PI OBR40_1.TIF, 20, 40, 10

Obr. 40.1.: Představa směrovače (a/) a brány (b/)

Další charakteristickou odlišností směrovačů od mostů je to, že jsou pro ostatní entity na úrovni síťové a linkové vrstvy viditelné. Mají své adresy, a pakety, které jimi mají projít, jim jsou explicitně adresovány (zatímco mosty zachycují veškerý provoz v každém z připojených segmentů). Proto také směrovače zpracovávají méně rámců než mosty, ovšem jejich zpracování je zase o to náročnější.

Je dobré si uvědomit, že pro funkci směrovače je nutné, aby vzájemně propojované sítě používaly stejný protokol na úrovni síťové vrstvy - podle něj totiž směrovač rozpoznává odesilatele i adresáta jednotlivých paketů, a rozhoduje o tom, kudy je dále odeslat. Není ovšem nutné, aby totéž platilo i na úrovni linkové a fyzické vrstvy. Zde se již konkrétní protokoly a přenosové technologie mohou lišit. Směrovače jsou dnes obvykle konstruovány tak, aby měly více různých rozhraní (tzv. portů), a bylo je možné vzájemně propojit například pomocí pevných okruhů, veřejných datových sítí, optických přenosových cest, a připojit k nim různé lokální sítě dle standardů IEEE 802 apod. Na obrázku 40.2. je pak dosti typický příklad možného propojení lokálních počítačových sítí ve čtyřech objektech (budovách). V rámci budov jsou jednotlivé segmenty připojeny na páteřní síť pomocí mostů, zatímco páteře jsou vzájemně spojeny prostřednictvím směrovačů (propojených optickým kabelem resp. pevným okruhem)

.PI OBR40_2.TIF, 20, 40, 10

Obr. 41.2.: Příklad propojení lokálních sítí pomocí mostů a směrovačů

Multiprotokolové směrovače

Požadavek stejného (a tudíž jediného) protokolu v síťové vrstvě je ovšem velmi omezující, zvláště v dnešní době, kdy vedle sebe koexistuje celá řada soustav protokolů (kromě ISO/OSI též TCP/IP, SNA, DECnet, SPX/IPX a další), a uživatelé volají po jejich co nejtěsnější integraci v rámci tzv. **heterogenních sítí** (tj. sítí, jejichž uzly používají různé soustavy protokolů).

Problém heterogenních sítí lze řešit v principu dvěma způsoby - konverzí protokolů, a směrováním více protokolů současně. Řešení prostřednictvím konverzí se ukázalo být značně náročné a nespolehlivé, a proto se prosadila především druhá možnost. Přední výrobci dnes nabízí tzv. **multiprotokolové směrovače** (**multiprotocol routers**), schopné pracovat současně s více různými protokoly. Multiprotokolový směrovač musí být schopen rozpoznat typ paketu, který dostane od linkové vrstvy, a

podle toho pak aplikovat ten směrovací algoritmus, který k příslušnému síťovému protokolu přísluší.

Brouter

V dnešní době, kdy dochází ke stále těsnějšímu propojování rozlehlých i lokálních sítí, je použití mostů i směrovačů velmi rozšířené. Rozhodnutí mezi tím, zda v určité situaci použít most či směrovač, nemusí být vždy okamžitě zřejmé, zvláště pak u lokálních sítí se složitější topologií a větším počtem používaných protokolů. V dnešní době však již existují také zařízení, která v sobě kombinují funkce obou těchto zařízení. V angličtině se pro jejich označení používá nejčastěji termín **bridge/router**, někdy též: **brouter**. Jde o zařízení, které se snaží fungovat jako směrovač, a teprve v okamžiku, kdy pro nějaký paket neumí aplikovat směrovací algoritmus, předá původní rámec dál tak, jako by to udělal most. Výhodou takovéhoto zařízení je pak i to, že se dokáže vyrovnat s takovými protokoly, které vůbec nelze směrovat (neboť nepočítají se síťovou vrstvou - jako například protokoly DECLAT (DEC Local Area Transport), LU 6.2 firmy IBM a protokoly NetBIOS).

Brána

Pokud je potřeba vzájemně propojit sítě zcela odlišných koncepcí, používající zcela jiné soustavy protokolů, je nutné použít propojovací zařízení, schopné provádět nezbytnou konverzi protokolů. Takovéto zařízení, označované nejčastěji jako **brána** (**gateway**, někdy též: **protocol converter**) pak pracuje na takové úrovni, na které je možné příslušnou konverzi zajistit - tedy například až na úrovni aplikační vrstvy, viz obrázek 40.1. b/.

Poznamenejme však ještě, že pojem "brána" resp. "gateway" se často používá i pro propojovací zařízení na nižších úrovních. Například v souvislosti s protokoly TCP/IP je termín "gateway" používán k označení směrovače (routeru).

41/ Vzájemné propojování sítí a RM ISO/OSI

V minulých dvou dílech našeho seriálu jsme se zabývali možností vzájemného propojování počítačových sítí na různých úrovních vrstvového síťového modelu. Ukažme si nyní, jak se k této problematice staví referenční model ISO/OSI.

Referenční model ISO/OSI samozřejmě počítá s možností vzájemného propojování jednotlivých sítí - veřejných i privátních, rozlehlých i lokálních - a v důsledku toho i s tím, že spojení mezi dvěma koncovými účastníky může ve skutečnosti procházet přes jednu či více mezilehlých sítí (které se v terminologii ISO označují jako **podsítě**, **subnetworks**). Pro transportní vrstvu, která zajišťuje přímou komunikaci koncových účastníků, by ale tento fakt neměl být vůbec viditelný - proto referenční model ISO/OSI svěřuje veškerou agendu, spojenou se vzájemným propojením podsítí, vrstvě síťové (viz též obrázek 41.1.).

.PI OBR41_1.TIF, 20, 40, 10

Obr. 41.1.: Představa vzájemného propojování sítí v referenčním modelu ISO/OSI

Síťová vrstva referenčního modelu ISO/OSI se pak člení na tři podvrstvy, a to (viz obr. 41.2.):

- podvrstvu přístupu k podsíti (**subnet access sublayer**), používající protokol **SNDAP** (**SubNetwork Dependent Access Protocol**),

- podvrstvu přizpůsobení podsítě (**subnet enhancement sublayer**), s protokolem **SNDCP** (**SubNetwork Dependent Convergence Protocol**),

- podvrstvu řízení vzájemně propojených podsítí (**internet sublayer**), s protokolem **SNICP** (**SubNetwork Independent Convergence Protocol**)

.PI OBR43_2.TIF, 20, 40, 10

Obr. 41.2.: Členění síťové vrstvy ISO/OSI na podvrstvy

Podvrstva přístupu k podsíti, která je z těchto tří podvrstev nejnižší, má na starosti to, co jsme až doposud přisuzovali celé síťové vrstvě jako takové - tedy především směrování jednotlivých paketů či datagramů - ovšem pouze v rámci jedné konkrétní podsítě (viz obrázek 41.3.), kde platí stejná pravidla pro směrování, kde se používají stejné adresy, kde jsou k dispozici stejné přenosové služby atd.

Sítě odlišného typu samozřejmě vyžadují různé podvrstvy přístupu, které ovšem poskytují nestejné služby, vyplývající z odlišného charakteru mechanismů a principů, na kterých jsou tyto sítě založeny. Jednotlivé podvrstvy přístupu k podsíti proto ještě není možné mezi sebou vzájemně propojit. Místo toho je nutné nejprve přizpůsobit jimi poskytované služby jednotnému standardu, což má za úkol prostřední ze tří zmíněných podvrstev, podvrstva přizpůsobení podsítě (viz opět obrázek 41.3.).

Pro nejvyšší podvrstvu se pak všechny podsítě "tváří" stejně. Podvrstva řízení vzájemně propojených sítí pak již může zajišťovat vše, co je potřeba k doručování jednotlivých paketů až k jejich konečnému příjemci - včetně jejich průchodu skrz jednu či několik mezilehlých sítí prostřednictvím existujících propojovacích zařízení (IS, Intermediary System).

.PI OBR41_3.TIF, 20, 40, 10

Obr. 41.3.: Průchod dat propojovacími zařízeními (IS, Intermediary System)

Pro správné pochopení smyslu prostřední podvrstvy, tedy podvrstvy přizpůsobení podsítě, je vhodné si poněkud upřesnit, v čem mohou spočívat odlišnosti jednotlivých sítí resp. podsítí, a jak se projevují.

Představme si jako příklad dvě lokální sítě, propojené prostřednictvím veřejné datové sítě na bázi doporučení X.25. Jedním z nejmarkantnějších rozdílů zde bude již samotná povaha přenosových služeb na úrovni síťové vrstvy. Zatímco lokální sítě budou nejspíše používat nespojované (connectionless) služby, síť na bázi X.25 používá spojovaný (connection-oriented) mechanismus virtuální okruhů. Při přenosu jednotlivých datagramů z jedné lokální sítě do druhé je pak nutné pro každý z nich vždy znovu zřídit nový virtuální okruh, a po přenesení datagramu jej zase ihned zrušit (případně ještě chvíli počkat, zda nebude záhy požadován přenos dalšího datagramu, pro který by se již vytvořený virtuální okruh dal ještě využít). Jde tedy vlastně o

emulaci mechanismu datagramové služby prostřednictvím mechanismu virtuálního okruhu, kterou zajišťuje právě podvrstva přizpůsobení podsítě.

Další možnou odlišností je maximální velikost datových paketů resp. rámců v jednotlivých podsítích. Například sítě na bázi X.25 umožňují, aby datový rámec měl až 32768 bitů, zatímco sítě typu Ethernet (resp. IEEE 802.3) připouští rámce jen do velikosti 12144 bitů, ale například sítě typu Token Bus (resp. IEEE 802.4) mohou přenášet rámce velikosti až 65528 bitů.

Problém nastává v okamžiku, kdy potřebujeme přenést pakety resp. rámce určité velikosti podsítí, která je schopna pracovat jen s menšími pakety resp. rámci - například při přenosu z jedné sítě Token Bus do jiné sítě Token Bus přes veřejnou datovou síť na bázi X.25. Zřejmým a v podstatě jediným možným řešením je rozdělit původní velký paket na několik menších paketů, tzv. **fragmentů**, a ty přenést jako samostatné celky. Otázkou ovšem je, kdo a kdy má provést zpětné sestavení jednotlivých fragmentů do původního celku. Zde jsou možné dva základní přístupy. První z nich, označovaný jako **transparentní fragmentace** (**transparent fragmentation**, někdy též: **intranet fragmentation**), předpokládá, že všechny fragmenty (dílčí pakety) jsou zpětně sestaveny do původního tvaru na výstupu z té podsítě, která fragmentaci vyvolala. Opačný přístup, označovaný jako **netransparentní fragmentace** (**nontransparent fragmentation**, někdy též: **internet fragmentation**), počítá naopak s tím, že jednotlivé fragmenty si do původního tvaru poskládá až jejich koncový příjemce. Oba přístupy ilustruje obrázek 41.4.

.PI OBR41_4.TIF, 20, 40, 10

Obr. 41.4. Představa fragmentace

a/ transparentní

b/ netransparentní

Další odlišnosti mezi jednotlivými podsítěmi se týkají například odlišného způsobu adresování a formátu adres, používaných v jednotlivých podsítích, mechanismu řízení toku dat a předcházení stavu zahlcení, zajišťování správy, hlášení nestandardních situací atd.

Podvrstva řízení vzájemně propojených sítí, nejvyšší ze tří podvrstev síťové vrstvy, může být koncipována tak, aby své bezprostředně vyšší (tj. transportní) vrstvě poskytovala služby buď spojovaného, nebo naopak nespojovaného charakteru. V současné době se však i v rámci referenčního modelu ISO/OSI, původně orientovaného výhradně na služby spojovaného charakteru, prosazuje spíše nespojovaná varianta. Organizace ISO přijala dokonce normu ISO 8473, definující protokol podvrstvy řízení vzájemně propojených podsítí (protokol SNICP, viz výše). Tento protokol, známý též jako **ISO Internet Protocol** či jen **ISO-IP**, totiž předpokládá právě nespojovaný charakter poskytovaných přenosových služeb. Je nezávislý na konkrétním charakteru podsítě, a je schopen zajistit jak transparentní, tak i netransparentní fragmentaci, směrování, řízení toku atd. Je do značné míry inspirován obdobným protokolem IP (Internet Protocol) ze soustavy protokolů TCP/IP.

42/ Síťový model TCP/IP

Referenční model ISO/OSI, kterým jsme se v našem seriálu dosud zabývali, je všeobecně považován za koncepci resp. síťovou architekturu, která do budoucna převládne a stane se dominující. Není ale zdaleka koncepcí jedinou, a své budoucí postavení si musí teprve vybojovat. V současné době je největším "rivalem" referenčního modelu ISO/OSI soustava protokolů TCP/IP.

Řekne-li se dnes TCP/IP, je to obvykle chápáno jen jako označení dvou přenosových protokolů, používaných v počítačových sítích s počítači na bázi Unixu, konkrétně protokolů TCP (Transmission Control Protocol) resp. IP (Internet Protocol). Ve skutečnosti ale zkratka TCP/IP označuje celou soustavu protokolů, ne nutně vázanou na operační systém Unix, přičemž TCP a IP jsou sice nejznámější protokoly této soustavy, ale zdaleka ne protokoly jediné. Správnější je ale považovat TCP/IP za ucelenou soustavu názorů o tom, jak by se počítačové sítě měly budovat, a jak by měly fungovat. Zahrnuje totiž i vlastní představu o tom, jak by mělo být síťové programové vybavení členěno na jednotlivé vrstvy, jaké úkoly by tyto vrstvy měly plnit, a také jakým způsobem by je měly plnit - tedy jaké konkrétní protokoly by na jednotlivých úrovních měly být používány. Ve smyslu definice, kterou jsme si zavedli ve 23. dílu našeho seriálu, je tedy TCP/IP síťovou architekturou.

Pohled do historie

Počátky TCP/IP se datují do konce 60. let, a jsou úzce spojeny s činností účelové agentury ARPA (Advanced Research Projects Agency) ministerstva obrany USA, která si nové protokoly nechala vyvinout pro svou počítačovou síť ARPANET. Na vývoji celé soustavy protokolů, financovaném prostřednictvím tzv. grantů ministerstva obrany (účelových dotací na výzkum) se pak podílela počítačově orientovaná pracoviště předních univerzit USA. Svou dnešní podobu získaly nové protokoly zhruba v letech 1977-79, a brzy poté na ně začala postupně přecházet i vlastní síť ARPANET, která se posléze stala zárodkem a páteří celého konglomerátu sítí, nazývaného dnes příznačně **Internet**.

Agentura ARPA (mezitím přejmenovaná na DARPA) se pak snažila prosadit nově vytvořené protokoly (a vlastně i celou ucelenou síťovou architekturu) do praktického života i mimo síť ARPANET, především pak do akademického prostředí. Většina univerzitních počítačových pracovišť v USA v té době provozovala nějakou verzi tzv. BSD Unixu, pocházející ze střediska Berkeley Software Distribution (BSD) na University of California v Berkeley. Agentura DARPA si proto nechala u přední americké firmy Bolt, Beranek and Newman (BBN) na zakázku vyvinout implementaci TCP/IP protokolů pod operační systém Unix, a univerzitě v Berkeley přispěla výzkumnými granty na začlenění těchto protokolů do univerzitou distribuovaných produktů. Tím se protokoly TCP/IP prosadily do BSD Unixu, a posléze pak i do ostatních verzí Unixu buď přímo jako jejich standardní součást, nebo jako volitelný doplněk (option). Díky své popularitě se však záhy dostaly i na jiné platformy, a dnes jsou implementovány snad ve všech výpočetních prostředích, od osobních počítačů PC s operačním systémem MS DOS až např. po sálové počítače (mainframes) firmy IBM a operační systém VM.

Pro svůj úzký vztah k síti Internet je soustava protokolů TCP/IP někdy označována také jako **Internet protocol suite** (doslova: soustava protokolů Internetu), nebo též **Department of Defense (DoD) protocol suite**, což naopak zdůrazňuje vztah k ministerstvu obrany USA.

Filosofie TCP/IP

Hlavní odlišnosti mezi referenčním modelem ISO/OSI a TCP/IP vyplývají především z rozdílných výchozích předpokladů a postojů jejich tvůrců. Jak jsme si již naznačili v předchozích dílech našeho seriálu, při koncipování referenčního modelu ISO/OSI měli hlavní slovo zástupci spojových organizací. Ti pak nově vznikajícímu modelu vtiskli svou vlastní představu - především spojovaný a spolehlivý charakter služeb, poskytovaných v komunikační podsíti (tj. až do úrovně síťové vrstvy, včetně). Jinými slovy: ISO/OSI model počítá se soustředěním co možná nejvíce funkcí, včetně zajištění spolehlivosti přenosů, již do komunikační podsítě, která v důsledku toho bude muset být poměrně složitá, zatímco k ní připojované hostitelské počítače budou mít relativně jednoduchou úlohu. Později se ale ukázalo, že například právě v otázce zajištění spolehlivosti to není nejšťastnější řešení - že totiž vyšší vrstvy nemohou považovat spolehlivou komunikační podsíť za dostatečně spolehlivou pro své potřeby, a tak se snaží zajistit si požadovanou míru spolehlivosti vlastními silami. V důsledku toho se pak zajišťováním spolehlivosti do určité míry zabývá vlastně každá vrstva referenčního modelu ISO/OSI.

Tvůrci protokolů TCP/IP naopak vycházeli z předpokladu, že zajištění spolehlivosti je problémem koncových účastníků komunikace, a mělo by tedy být řešeno až na úrovni transportní vrstvy. Komunikační podsíť pak podle této představy nemusí ztrácet část své přenosové kapacity na zajišťování spolehlivosti (na potvrzování, opětné vysílání poškozených paketů atd.), a může ji naopak plně využít pro vlastní datový přenos. Komunikační podsíť tedy podle této představy nemusí být zcela spolehlivá - může v ní docházet ke ztrátám přenášených paketů, a to bez varování a bez snahy o nápravu. Komunikační síť by ovšem neměla zahazovat pakety bezdůvodně. Měla by naopak vyvíjet maximální snahu přenášené pakety doručit (v angličtině se v této souvislosti používá termín: **best effort**), a zahazovat pakety až tehdy, když je skutečně nemůže doručit - tedy např. když dojde k jejich poškození při přenosu, když pro ně není dostatek vyrovnávací paměti pro dočasné uložení, v případě výpadku spojení apod. Na rozdíl od referenčního modelu ISO/OSI tedy TCP/IP předpokládá jednoduchou (ale rychlou) komunikační podsíť, ke které se připojují inteligentní hostitelské počítače.

Další odlišnost od referenčního modelu ISO/OSI spočívá v názoru na to, jak má komunikační síť vlastně fungovat. Zatímco model ISO/OSI počítá především se spojovaným přenosem - tedy s mechanismem virtuálních okruhů, TCP/IP naopak předpokládá nespojovaný charakter přenosu v komunikační podsíti - tedy jednoduchou datagramovou službou - což ostatně vyplývá i z představy co možná nejjednodušší komunikační podsítě.

Čtyři vrstvy TCP/IP

Zatímco referenční model ISO/OSI vymezuje sedm vrstev síťového programového vybavení, TCP/IP počítá jen se čtyřmi vrstvami - viz obrázek 42.1.

.PI OBR42_1.TIF, 20, 40, 10

Obr. 42.1.: Čtyři vrstvy síťového modelu TCP/IP vs. RM ISO/OSI

Nejnižší vrstva, **vrstva síťového rozhraní** (**Network Interface Layer**) (někdy též: **linková vrstva** resp. **Link Layer**) má na starosti vše, co je spojeno s ovládáním konkrétní přenosové cesty resp. sítě, a s přímým vysíláním a příjmem datových paketů. V rámci soustavy TCP/IP není tato vrstva blíže specifikována, neboť je závislá na použité přenosové technologii.

Vrstvu síťového rozhraní může tvořit relativně jednoduchý ovladač (device driver), je-li daný uzel přímo připojen například k lokální síti či ke dvoubodovému spoji, nebo může tato vrstva představovat naopak velmi složitý subsystém, s vlastním linkovým přenosovým protokolem (např. HDLC apod.). Vzhledem k velmi častému připojování jednotlivých uzlů na lokální sítě typu Ethernet je vrstva síťového rozhraní v rámci TCP/IP často označována také jako **Ethernetová vrstva** (**Ethernet Layer**).

Bezprostředně vyšší vrstva, která již není závislá na konkrétní přenosové technologii, je vrstva **síťová**, v terminologii TCP/IP označovaná jako **Internet Layer** (volněji: vrstva vzájemného propojení sítí), nebo též **IP vrstva** (**IP Layer**) podle toho, že je realizována pomocí protokolu IP. Úkol této vrstvy je v prvním přiblížení stejný, jako úkol síťové vrstvy v referenčním modelu ISO/OSI - stará se o to, aby se jednotlivé pakety dostaly od odesilatele až ke svému skutečnému příjemci, přes případné směrovače resp. brány. Vzhledem k nespojovanému charakteru přenosů v TCP/IP je na úrovni této vrstvy zajišťována jednoduchá (tj. nespolehlivá) datagramová služba.

Třetí vrstva TCP/IP je označována jako **transportní vrstva** (**Transport Layer**), nebo též jako **TCP vrstva** (**TCP Layer**), neboť je nejčastěji realizována právě protokolem TCP (Transmission Control Protocol). Hlavním úkolem této vrstvy je zajistit přenos mezi dvěma koncovými účastníky, kterými jsou v případě TCP/IP přímo aplikační programy (jako entity bezprostředně vyšší vrstvy). Podle jejich nároků a požadavků může transportní vrstva regulovat tok dat oběma směry, zajišťovat spolehlivost přenosu, a také měnit nespojovaný charakter přenosu (v síťové vrstvě) na spojovaný.

Přestože je transportní vrstva TCP/IP nejčastěji zajišťována právě protokolem TCP, není to zdaleka jediná možnost. Dalším používaným protokolem na úrovni transportní vrstvy je například protokol UDP (User Datagram Protocol), který na rozdíl od TCP nezajišťuje mj. spolehlivost přenosu - samozřejmě pro takové aplikace, které si to (na úrovni transportní vrstvy) nepřejí.

Nejvyšší vrstvou TCP/IP je pak vrstva **aplikační** (**Application Layer**). Jejími entitami jsou jednotlivé aplikační programy, které na rozdíl od referenčního modelu ISO/OSI komunikují přímo s transportní vrstvou. Případné prezentační a relační služby, které v modelu ISO/OSI zajišťují samostatné vrstvy, si zde musí jednotlivé aplikace v případě potřeby realizovat samy.

43/ TCP/IP a vzájemné propojování sítí

Způsob, jakým je v referenčním modelu ISO/OSI řešena otázka vzájemného propojování sítí, jsme si naznačili ve 41. dílu našeho seriálu. Z nejrůznějších materiálů o referenčním modelu však lze vycítit, že tato problematika do něj byla zapracována spíše až na poslední chvíli, a rozhodně nepatří mezi nejsilnější stránky celého referenčního modelu ISO/OSI. V případě TCP/IP je tomu jinak - vzájemné propojování sítí stejného i odlišného typu bylo jedním z výchozích

předpokladů celé soustavy protokolů TCP/IP, a je v ní také řešeno mnohem systematičtěji, než v referenčním modelu ISO/OSI.

Filosofie TCP/IP od začátku usiluje o co nejuniverzálnější propojení sítí různých typů - od lokálních sítí typu Ethernet, Token Ring apod., přes veřejné datové sítě až po rozlehlé sítě celosvětového dosahu. Klade si přitom za cíl umožnit každému počítači komunikovat s kterýmkoli jiným počítače, bez ohledu na to, zda mezi nimi existuje přímé spojení, nebo zda jsou například oba uzly různých sítí, které jsou vzájemně propojeny jednou nebo několika dalšími sítěmi. Výsledkem je pak jediná soustava vzájemně propojených sítí, v terminologii TCP/IP označovaná obecně jako **internet** (s malým "i", viz též 39. díl našeho seriálu). Z pohledu uživatele by však vnitřní struktura této

.PI OBR43_1.TIF, 20, 40, 10

Obr. 43.1: Vzájemně propojené sítě v síťovém modelu TCP/IP

a/ z pohledu uživatele (resp. aplikační a transportní vrstvy)

b/ skutečná topologie

soustavy sítí měla být irelevantní - uživatelé, resp. jejich aplikační programy, se mohou na celý internet dívat (podle obrázku 43.1 a/) jako na jedinou velkou síť, ke které jsou připojeny jednotlivé koncové počítače - v terminologii TCP/IP označované jako **hostitelské počítače** (**host computers, hosts**).

Brány, čili IP směrovače

Ve skutečnosti je ovšem výsledná soustava - internet - stále jen konglomerátem (dílčích) sítí stejného či různého typu, vzájemně propojených pomocí propojovacích zařízení**.** Tato propojovací zařízení se přitom v terminologii TCP/IP označují jako **brány** (**gateway**), viz obrázek 43.1 b/. Podobně jako referenční model ISO/OSI, předpokládají i protokoly TCP/IP, že propojení jednotlivých dílčích sítí bude realizováno na úrovni síťové vrstvy. Podle obvyklých definic, které jsme si uvedli ve 40. dílu našeho seriálu, jsou pak ale brány (v terminologii TCP/IP) ve skutečnosti spíše směrovači (routers). Tento dosti nepříjemný terminologický konflikt se některé novější odborné prameny snaží řešit zavedením výstižnějšího termínu **IP router** (tj. IP směrovač).

Úloha síťové vrstvy

Je to tedy právě síťová vrstva (resp. tzv. IP vrstva, realizovaná protokolem IP, viz minule), která v síťovém modelu TCP/IP zajišťuje potřebné směrování mezi jednotlivými dílčími sítěmi, a vyšším vrstvám tak vytváří iluzi jediné homogenní sítě dle obrázku 43.1 a/. Sama však musí pracovat se skutečnou vnitřní strukturou resp. způsobem vzájemného propojení (sama tedy pracuje s představou dle obrázku 43.1 b/).

Síťová vrstva se však musí vyrovnávat i s konkrétními odlišnostmi jednotlivých dílčích sítí - například s odlišným charakterem adres, s různou maximální velikostí přenášených paketů resp. rámců a jejich formátem, s odlišným charakterem poskytovaných přenosových služeb (spojovaných či nespojovaných) apod. Pro každou síť či každý přenosový kanál, na který je brána připojena, má samostatný ovladač na úrovni vrstvy síťového rozhraní (viz obrázek 43.2.).

Ovladač na úrovni vrstvy síťového rozhraní dokáže odstínit síťovou vrstvu od konkrétního způsobu ovládání příslušné sítě a přesného formátu datových rámců. Není ovšem již v jeho silách zastřít před síťovou vrstvou rozdíl v používaném mechanismu adresování, resp. zajistit používání jednotných adres ve všech dílčích sítích. Tento jednotný způsob adresování může zajistit až síťová vrstva.

.cp9

.PI OBR43_2.TIF, 20, 40, 10

Obr. 43.2.: Představa brány a ovladačů na úrovni vrstvy síťového rozhraní

Jednotná abstrakce

Síťová vrstva, resp. protokol IP, který se pro její realizaci používá, vytváří jednotnou abstrakci všech dílčích sítí, která umožňuje pracovat s jediným typem virtuální sítě, namísto s více či méně odlišnými dílčími sítěmi. Tato jednotná abstrakce spočívá nejen v zavedení jednotného způsobu adresování, ale také v jednotném formátu datových paketů, používaných na úrovni síťové vrstvy, tzv. **IP datagramů** (**IP datagrams**), a v poskytování jednotné přenosové služby - nespolehlivé nespojované (datagramové) služby.

IP adresy

Adresy, které protokol IP zavádí (a které jsou proto označovány jako tzv.**IP adresy**), jsou 32-bitové. Z pohledu transportní a aplikační vrstvy je lze interpretovat jako lineární (resp. jednosložkové) adresy - což odpovídá představě jediné homogenní sítě dle obrázku 43.1. a/. Na úrovni síťové vrstvy se ale interpretují jako dvousložkové, tvořené číslem resp. adresou hostitelského počítače, a číslem resp. adresou (dílčí) sítě, ve které se tento hostitelský počítač nachází (tato interpretace pak odpovídá představě vzájemně propojených dílčích sítí dle obrázku 43.1. b/). Přesnému formátu a významu IP adres se budeme ještě věnovat v samostatném dílu našeho seriálu.

Je dobré si znovu zdůraznit, že IP adresy jsou pouze abstraktními adresami, které musí být posléze převedeny na skutečné fyzické adresy. Kdykoli totiž ovladač na úrovni vrstvy síťového rozhraní dostane nějaká data k odeslání, musí spolu s nimi dostat i konkrétní fyzickou adresu, na kterou je má odeslat. Sám totiž s IP adresami nepracuje. Jde-li například o lokální síť typu Ethernet, dostane síťová vrstva (od vrstvy transportní) adresu cílového hostitelského počítače ve formě 32-bitové IP adresy, ale příslušný ovladač vrstvy síťového rozhraní musí dostat 48-bitovou Ethernetovou adresu. K mechanismu, kterým se v TCP/IP sítích zjišťuje korespondence mezi IP adresami a konkrétními specifickými adresami, se samozřejmě vrátíme ještě podrobněji.

IP datagramy

Podobně jako jednotný formát adres a způsob adresování, zavádí protokol IP i jednotný formát přenášených dat na úrovni síťové vrstvy - již výše zmíněné IP datagramy. Jde o datové pakety, označované jako datagramy proto, že jsou přenášeny pomocí nespojované (též: datagramové) síťové přenosové služby.

Na úrovni vrstvy síťového rozhraní jsou tyto datagramy přenášeny pomocí takových rámců, se kterými příslušná dílčí síť pracuje. Tyto se ovšem v obecném případě síť od sítě liší!

.PI OBR43_3.TIF, 20, 40, 10

Obr. 43.3.: Přenos dat při existenci brány mezi sítěmi

.cp9

Význam přenášených dat na úrovni jednotlivých vrstev ilustruje obrázek 34.3.: zatímco aplikační a transportní vrstvy vůbec "neví" o existenci jednotlivých dílčích sítí (a jako koncoví účastníci přenosu si tak předávají vždy přesně stejný tvar paketů, resp. zpráv), na úrovni vrstvy síťového rozhraní se přenáší identické rámce jen v jednotlivých dílčích sítí. V každé z nich je pak přenášen IP datagram vždy stejného formátu, který se ovšem přeci jen v něčem liší - například v hodnotě čítače, který vyjadřuje životnost paketu (což je součást mechanismu, který má zabránit případnému zacyklení).

44/ Adresování v TCP/IP sítích - I.

V minulém dílu našeho seriálu jsme si naznačili, že síťový model TCP/IP zavádí na úrovni síťové vrstvy jednotnou abstrakci všech dílčích sítí, které tvoří výslednou soustavu vzájemně propojených sítí - tzv. internet. Podstata této abstrakce spočívá mj. i v použití jednotných 32-bitových adres, a to bez ohledu na skutečné (fyzické) adresy, používané v jednotlivých dílčích sítích. Podívejme se nyní na otázku IP adres poněkud podrobněji.

Jak jsme si již také uvedli minule, 32-bitové adresy, používané na úrovní síťové vrstvy (tzv. IP adresy), mohou být chápány jako jednosložkové (lineární) adresy, vyjadřující právě a pouze adresu jednoho hostitelského počítače. Pro síťovou vrstvu je ale 32-bitová IP adresa vždy dvousložková, tvořená číslem resp. adresou (dílčí) sítě a číslem resp. adresou hostitelského počítače, který se v této síti nachází- viz obr. 44.1. a/. Tato představa pak odpovídá členění výsledné soustavy sítí (internet-u) na jednotlivé dílčí sítě (viz minule, obr. 43.1. a/), kterými mohou být například jednotlivé lokální sítě typu Ethernet, Token Ring apod., až po velké rozlehlé sítě s velkým počtem hostitelských počítačů.

.PI OBR44_1.TIF, 20, 40, 10

Obr. 44.1.: Představa IP adresy

a/ bez členění na podsítě

b/ se členěním na podsítě (tzv. subnetting)

.cp9

Podstata směrování v TCP/IP sítích

Důvod, proč síťová vrstva pracuje právě s takovouto představou, a nikoli s představou jednolité, dále nestrukturované výsledné sítě, je veskrze praktický: v zájmu minimalizace objemu směrovacích tabulek je směrování v TCP/IP sítích

založeno jen na adresách (dílčích) sítí, a nikoli na adresách jednotlivých hostitelských počítačů v rámci těchto sítí.

Právě vyslovené tvrzení si zaslouží poněkud upřesnit: Každý hostitelský počítač, který chce odeslat nějaký IP datagram jinému hostitelskému počítači, dokáže z IP adresy příjemce snadno rozpoznat, zda tento leží ve stejné dílčí síti či nikoli. Pokud ano (nachází-li se například v téže síti typu Ethernet), pošle mu odesilatel svůj datagram přímo. Pokud se ale příjemce nachází v jiné síti, pošle odesilatel svůj datagram nejbližší bráně (resp. IP směrovači, viz minule) ve "své" síti. Na ní je pak rozhodnout, kudy datagram poslat dále. Podstatné přitom je, že při svém rozhodování vychází brána pouze z té části IP adresy konečného příjemce, která vyjadřuje příslušnou cílovou síť. V prvním přiblížení si lze představit, že každá brána má své směrovací tabulky ve formě seznamu dvojic <*síť,brána*>, a podle cílové sítě příjemce si v nich najde, které bráně má příslušný datagram poslat dále. Zbývající část IP adresy příjemce, která vyjadřuje adresu hostitelského počítače v rámci cílové sítě, pak využije až ta brána (poslední v řetězci), která již leží v příslušné cílové síti, a která pak datagram pošle přímo jeho konečnému adresátovi.

Třídy IP adres

32 bitů, vyhrazených pro IP adresy, je tedy nutné vhodným způsobem rozdělit na dvě části, tak aby tyto mohly vyjadřovat adresu sítě a adresu hostitelského počítače v rámci této sítě. Otázkou ovšem je, jak toto rozdělení provést. Filosofie síťového modelu TCP/IP totiž předpokládá, že jednotlivými dílčími sítěmi mohou být jak "malé" sítě s několika málo uzly (hostitelskými počítači), tak i "velké" sítě s tisíci i desítkami tisíc uzlů. Třicet dva bitů, které jsou k dispozici, pak nelze rozdělit jediným způsobem, který by pamatoval na sítě s velkým počtem uzlů, a současně měl i dostatečně velkou rezervu pro rychle rostoucí počet jednotlivých dílčích sítí.

Řešení, uplatněné u IP adres, pak spočívá v zavedení tří různých způsobů rozdělení 32 bitů na dvě části, resp. v zavedení tří různých formátů IP adres: jednoho pro dílčí sítě s velkým počtem uzlů resp. hostitelských počítačů, jednoho pro "středně velké" sítě, a jednoho pro "malé" sítě, s relativně malým počtem uzlů. Tyto formáty, označované jako **třídy A, B** a **C** (**class A, B** and **C**), ukazuje obrázek 44.2.

.PI OBR44_2.TIF, 20, 40, 10

Obr. 44.2.: Třídy IP adres

Adresy třídy A mají pro adresu sítě vyhrazeno 7 bitů, a pro adresu hostitelského počítače 24 bitů. Adres této třídy je tedy relativně velmi málo, počítají však s velmi velkým počet uzlových počítačů. Jsou určeny pro velké rozlehlé sítě, jako např. síť ARPAnet (viz 39. díl seriálu), která se stala zárodkem sítě Internet, pro velké veřejné sítě, případně i velké podnikové sítě. Adresy třídy B, které připouští až 65534 uzlů (hostitelských počítačů), jsou určeny pro středně velké sítě, např. pro univerzitní sítě, podnikové sítě apod., a používají se též v souvislosti s tzv. subnetting-em (podsítěmi, viz dále). Adresy třídy C, které počítají jen se 254 uzly, jsou pak určeny pro všechny ostatní sítě.

Symbolický zápis IP adres

32-bitové IP adresy je samozřejmě možné vyjadřovat jako celá dvojková čísla. Pro člověka to ale není příliš srozumitelné, a tak se pro symbolický zápis IP adres zavedla vhodná konvence, označovaná jako **tečkovaná desítková notace** (**dotted decimal**

notation). Spočívá v tom, že 32 bitů IP adresy se rozdělí na čtyři části po osmi bitech (oktety), a každá z těchto částí se pak vyjádří jako celé desítkové číslo bez znaménka (s použitím tečky jako oddělovače jednotlivých částí). Například nepříliš mnemonický tvar 10000000 00000001 00000010 00000011 tak dostává přehlednější podobu: 128.1.2.3. Význam desítkové notace ilustruje též tabulka 44.3.

	první adresa oktet sítě*1	adresa hostitelského počítače*1	maximální počet hostitelských počítačů
třída A	1-126	p	q.r.s	16777214
třída B	128-191	p.q	r.s	65534
třída C	192-223	p.q.r.	s	254

*+Je-li p.q.r.s obecný tvar IP adresy

Tabulka 44.3.: Desítková tečkovaná notace a IP adresy

Jak volit IP adresy?

V principu je možné volit IP adresy dle vlastního uvážení. Je ovšem nutné dbát na dodržení jedné zásadní podmínky, a tou je jednoznačnost. Nesmí se totiž stát, aby se v jakémkoli konglomerátu vzájemně propojených sítí adresy dvou různých sítí shodovaly (přesněji ty části IP adres, které vyjadřují adresu sítě) - jinak totiž bude mít mechanismus směrování velké problémy.

U každé počítačové sítě či soustavy sítí, budované pomocí protokolů TCP/IP, je však prakticky jen otázkou času, kdy je jejich provozovatelé budou chtít připojit do dnes již celosvětové sítě Internet (viz např. CW 7/92). Také zde samozřejmě platí zásada jednoznačnosti adres, takže všechny nově připojované sítě musí mít dosud nepoužité adresy. V Internetu proto existuje centrální autorita (konkrétně: DDN Network Information Center, SRI International, 333 Rawenswood Avenue, Menlo Park, California 94025), která se stará o hospodaření s IP adresami, a přiděluje je všem potenciálním zájemcům {přesněji: přiděluje tu část IP adres, která představuje adresu sítě, zatímco zbývající část IP adres si volí žadatel. Ve skutečnosti tak přiděluje vlastně celé skupiny IP adres se shodnou první částí). Skutečné připojení k síti Internet přitom vůbec není podmínkou pro přidělení adresy, takže je naopak dobré si přidělení adresy vyžádat co nejdříve, pokud možno ještě před zprovozněním jakékoli sítě na bázi TCP/IP (a tím se vyhnout pozdější změně IP adres, která je samozřejmě možná, ale obvykle značně pracná).

V našich zeměpisných šířkách však není nutné se pro přidělení IP adresy obracet až do USA. DDN NIC, která adresy spravuje, totiž delegovala svou pravomoc v přidělování IP adres pro oblast ČSFR výpočetnímu centru pražské VŠCHT (Technická 5, 166 28 Praha 6).

.cp20

Subnetting

Jak jsme si již naznačili výše, při směrování v TCP/IP sítích se vychází pouze z té části IP adresy, která představuje adresu (dílčí) sítě. Výhodou jsou neskonale menší směrovací tabulky, než jaké by musely být v případě směrování podle celých IP adres.

Přesto se ale v síti Internet záhy ukázalo, že i tak vychází směrovací tabulky větší, než by bylo vhodné. Proto se zpětně do TCP/IP prosadil mechanismus, označovaný v angličtině jako **subnetting**, který se snaží další nárůst směrovacích tabulek omezit.

.PI OBR44_4.TIF, 20, 40, 10

Obr. 44.4.: Představa členění na podsítě (tzv.subnetting)

Myšlenku tohoto mechanismu si můžeme ukázat na příkladu organizace, připojené k síti Internet, která provozuje více dílčích sítí, a pro každou z nich má přidělenu samostatnou adresu (nejspíše třídy C). Každá z těchto dílčích sítí je pak ovšem "samostatně viditelná" i z vně příslušné organizace, a jako se samostatnou je s ní také nakládáno - ve směrovacích tabulkách každá zabírá samostatnou položku. Kdyby se ovšem příslušná organizace sama postarala o potřebné směrování mezi svými sítěmi, a navenek vystupovala jako jediný celek, mohla by mít přidělenu jen jednu síťovou adresu (například třídy B místo více síťových adres třídy C), a ve směrovacích tabulkách by tak zabírala jen jednu položku. Všechny dílčí sítě, patřící příslušné organizaci, by tak navenek vystupovaly jako síť jediná, s jedinou společnou adresou (přesněji: s IP adresami, které se shodují v části, vyjadřující adresu sítě). Skutečné vnitřní členění této navenek jednotné sítě by pak bylo plně v kompetenci příslušné organizace. Ta by také musela změnit přesný způsob, jakým sama interpretuje IP adresy svých hostitelských počítačů, a to podle obrázku 44.1. b/ - tu část, která z vnějšího pohledu představuje číslo resp. adresu hostitelského počítače, by pro sebe musela rozdělit na dvě části - číslo resp. adresu své dílčí sítě (podsítě, neboli subnet, odsud **subnetting**), a na číslo resp. adresu hostitelského počítače v rámci této podsítě. V nejjednodušším případě adresy třídy B tak, že třetí oktet bude považovat za číslo podsítě, a čtvrtý za číslo uzlu (hostitelského počítače) - viz obrázek 44.4. V obecném případě pak podle tzv. **masky podsítě** (**subnet mask**), viz obrázek 44.1. a/, která obě části IP adresy nově definuje.

45/ Adresování v TCP/IP sítích - II.

V minulém dílu našeho seriálu jsme se podrobněji zabývali IP adresami. Víme již, že představují jednotné adresy, které používá libovolný konglomerát vzájemně propojených sítí na bázi soustavy protokolů TCP/IP. Jsou však stále jen abstrakcí na úrovni síťové vrstvy, která odpovídá představě jednotné virtuální sítě. Ta je ale ve skutečnosti realizována dílčími sítěmi více či méně odlišného typu, které používají své vlastní mechanismy adresování a formáty adres. Proto také IP adresy musí být převáděny na takové konkrétní (fyzické) adresy, jaké příslušná dílčí síť skutečně používá (na úrovni vrstvy síťového rozhraní). Otázkou ovšem je, jak takovýto převod vůbec realizovat.

Představme si dva hostitelské počítače A a B, které mají pořadě IP adresy I_A a I_B. Předpokládejme dále, že jde o uzly téže (dílčí) sítě, které díky tomu mohou komunikovat mezi sebou přímo (resp. nejsou odkázány na bránu, resp. IP směrovač, propojující různé dílčí sítě). V rámci "své" dílčí sítě přitom mají oba uzly fyzické adresy F_A a F_B. Jestliže nyní síťová vrstva (IP vrstva) počítače A dostane od své transportní vrstvy za úkol přenést určitá data počítači s IP adresou I_B (tj. počítači B), musí být schopna zajistit převod IP adresy I_B na fyzickou adresu F_B. Tu totiž potřebuje příslušný ovladač v bezprostředně nižší vrstvě (vrstvě síťového rozhraní), aby mohl přenášená data skutečně doručit. Podobně počítač B: jakmile bude chtít počítači A odpovědět, musí vědět, jaká fyzická adresa (F_A) odpovídá IP adrese počítače A (I_A).

Problém transformace adres a jeho řešení

Problém transformace adres vyšší úrovně na adresy nižší úrovně, konkrétně nalezení odpovídající fyzické adresy k IP adrese, se označuje jako **address resolution problem**. Je možné jej řešit například formou tabulky, obsahující seznam vzájemně si odpovídajících adres. Je to ovšem spojeno s četnými problémy - kdo a jak zajistí počáteční naplnění tabulky, kdo ji bude udržovat a přizpůsobovat momentálnímu stavu sítě, kdo zajistí, aby její velikost nepřesáhla únosnou mez atd.

Tam, kde je to jen trochu možné, se proto používají spíše jinářešení, která jsou ovšem závislá na konkrétní povaze (dílčí) sítě a jím používaném mechanismu adresování.

Řešení pomocí přímého převodu

Velmi jednoduchá myšlenka, které se v této souvislosti sama nabízí, je řešit převod výčtem (tj. pomocí tabulky), ale pomocí vhodné tranformační funkce (vzorečku pro převod). To je ale možné jen tam, kde si uživatel resp. zřizovatel sítě může fyzické adresy jednotlivých uzlových počítačů volit sám, podle vlastních potřeb. Tak je tomu například v sítích ARCnet či proNET-10, kde si uživatel při instalaci síťové karty do svého počítače sám volí její fyzickou adresu (a tím i fyzickou adresu počítače, připojeného prostřednictvím této síťové karty). Má-li například volitelná fyzická adresa rozsah 8 bitů (jako je tomu právě u sítí ARCnet i proNET-10), je nejjednodušší volit ji shodně s posledním oktetem (posledními osmi bity) IP adresy. Transformace IP adresy na fyzickou se pak stává zcela triviální.

Řešení pomocí dynamické vazby

Možnost volit fyzickou adresu přímo na síťovém adaptéru při jeho instalaci je v praxi únosná jen pro adresy malého rozsahu. Především je ale spojena s potenciálním nebezpečím lidských chyb, které mohou vyústit v existenci dvou adaptérů resp. uzlů se stejnou fyzickou adresou v jedné síti. Jiné síťové technologie se proto k celému problému staví opačně - uživateli nedávají žádnou možnost ovlivnit fyzickou adresu síťového adaptéru. Ten ji má v sobě jednou provždy pevně zabudovánu.

Takto je tomu například u lokálních sítí typu Ethernet. Ty používají fyzické adresy v rozsahu 48 bitů, které v příslušných síťových adaptérech nastavuje přímo jejich výrobce. Aby se zajistila jednoznačnost adres i mezi síťovými adaptéry různých výrobců, musí si každý z nich nechat přidělit určitý rozsah adres od centrální autority,

která Ethernetovské adresy spravuje (a kterou je v tomto konkrétním případě americké sdružení IEEE).

Jakmile je ale potřeba transformovat 32-bitové IP adresy na 48-bitové Ethernetovské (či jiné "velké" adresy, které není možné podle potřeby volit), nezbývá než vrátit se zpět k převodním tabulkám, definujícím vzájemnou vazbu mezi jednotlivými adresami. Pokud možno ale nikoli ke statickým tabulkám, ale naopak k tabulkám dynamickým, které se vytváří a modifikují průběžně, podle okamžitého stavu sítě.

Protokol ARP

V soustavě protokolů TCP/IP je zahrnut velmi elegantní mechanismus dynamického budování a udržování převodních tabulek mezi IP adresami a fyzickými adresami, založený na protokolu **ARP** (**Address Resolution Protocol**). Ten využívá schopnosti všesměrového vysílání (tzv. broadcastingu) v některých sítích, které umožňují adresovat datový rámec všem uzlům dané lokální (resp. dílčí) sítě současně - bez nutnosti znát jejich konkrétní adresy. Například v sítích typu Ethernet lze vyslat datový rámec na jednu, předem známou speciální adresu, na kterou "slyší" všechny síťové adaptéry bez ohledu na svou konkrétní fyzickou adresu. Protokol ARP této možnosti využívá tak, že si jejím prostřednictvím nechá najít majitele příslušné IP adresy:

Představme si situaci, kdy jeden uzlový počítač chce zaslat nějaká data jinému počítači v téže dílčí síti. Zná však pouze jeho IP adresu, nikoli jeho fyzickou adresu. Protokol ARP prvního počítače proto využije možnosti všesměrového vysílání, a všem uzlům dané dílčí sítě pošle zvláštní rámec resp. paket s dotazem: "Kdo má IP adresu ....?" (viz obr. 45.1. a/). Tento rámec přijmou všechny uzly, a všechny také vyhodnotí paket, který je v něm obsažen. Pouze uzel B však rozpozná, že obsahuje jemu určený dotaz, a tak na něj odpoví zasláním své fyzické adresy (opět prostřednictvím speciálního paketu, jehož formát definuje protokol ARP). Ostatní uzly přitom na původní dotaz neodpovídají - viz obr. 45.1. b/.

.PI OBR45_1.TIF, 20, 40, 10

Obr. 45.1.: Zjištění fyzické adresy podle IP adresy (protokol ARP)

Nebylo by ale únosné se takovýmto způsobem ptát při každém jednotlivém přenosu vždy znovu. Každý uzlový počítač si proto sám průběžně vytváří potřebnou převodní tabulku mezi IP adresami a fyzickými adresami (ve vhodné vyrovnávací paměti), a právě naznačený mechanismus využívá až v případě, kdy ji potřebuje doplnit či aktualizovat.

.cp20

Protokol RARP

Protokol ARP umožňuje, aby každý hostitelský počítač (uzel) po svém spuštění vystačil jen se znalostí své vlastní fyzické adresy a své vlastní IP adresy (kterou si obvykle přečte z konfiguračního souboru na svém pevném disku). Na fyzické adresy všech ostatních uzlů ve své dílčí sítí se pak vhodně "doptá". Otázkou ovšem je, jak

tomu bude v případě bezdiskových stanic, které si svou IP adresu z vlastního pevného disku přečíst nemohou.

Po svém spuštění si každá bezdisková stanice musí svou vlastní IP adresu nejprve vyžádat na jiném uzlovém počítači, který vůči ní vystupuje v roli serveru IP adres. Způsob, jakým se na něj bezdisková stanice obrací, je analogický výše naznačenému mechanismu protokolu ARP - prostřednictvím všesměrového vysílání bezdisková stanice rozešle všem ostatním uzlům dotaz typu: "Jaká je moje IP adresa?". Sebe sama přitom stanice identifikuje prostřednictvím fyzické adresy, kterou má zabudovánu ve svém síťovém adaptéru - viz obrázek 45.2. a/.

.PI OBR45_2.TIF, 20, 40, 10

Obr. 45.2.: Zjištění vlastní IP adresy pro bezdiskovou stanici (protokol RARP)

Konkrétní protokol, prostřednictvím kterého si bezdisková stanice může svou IP adresu vyžádat, vychází z protokolu ARP, a je označován jako RARP (Reverse Address Resolution Protocol).

46/ Směrování v TCP/IP sítích - I.

Další z aspektů, důležitých pro pochopení celkové filosofie soustavy protokolů TCP/IP, je otázka směrování ve vzájemně propojených sítích. Z ní jsme si něco naznačili již ve 44. dílu našeho seriálu, nyní se však touto problematikou budeme zabývat podrobněji.

Nejprve si ale připomeňme, jakým způsobem se protokoly TCP/IP dívají na vzájemně propojené sítě (tzv. internet): předpokládají, že jednotlivé (dílčí) sítě jsou propojeny prostřednictvím **bran** (**gateways**), označovaných také jako **IP směrovače** (**IP routers**) - viz obrázek 46.1. (srovnej s obrázkem 43.1.). Vzájemné propojení je přitom takové, že mezi libovolnými dvěma dílčími sítěmi vždy existuje alespoň jedna cesta, která ovšem může vést i přes více jiných dílčích sítí, resp. procházet posloupností bran, propojujících mezilehlé sítě.

.PI OBR46_1.TIF, 20, 40, 10

Obr. 46.1.: Představa vzájemně propojených sítí

Každá brána je vždy připojena nejméně do dvou dílčích sítí, a slouží pouze potřebám směrování (a nikoli k provozování uživatelských aplikací). Tím se brány odlišují od druhého typu uzlů, které naopak slouží především k provozování aplikačních programů, a které se v terminologii TCP/IP označují jako **hostitelské počítače** (**hosts, host computers**). Také hostitelské počítače však mohou být připojeny do dvou či více dílčích sítí současně (pak jde o tzv. **multi-homed hosts**), a mohou tedy fungovat i jako brány. Přestože se toto řešení v praxi občas používá (hlavně v akademickém prostředí), není vždy bezproblémové. Filosofie TCP/IP však velmi ostře rozlišuje mezi hostitelským počítačem a bránou, a proto i my se přidržíme představy dvou fyzicky různých zařízení.

Přímé a nepřímé směrování

Obecně lze říci, že na směrování jakožto rozhodování o tom, kudy dále poslat datový paket, se podílí oba druhy uzlů TCP/IP sítí, tedy jak brány, tak i hostitelské počítače. Záměr je ovšem takový, aby se hostitelské počítače zabývaly směrováním jen v minimální možné míře, a maximum práce na tomto poli přenechaly bránám.

Představme si situaci na obrázku 46.2., kdy hostitelský počítač A chce odeslat IP datagram hostitelskému počítači B. Z IP adresy počítače B jednoduchým způsobem rozpozná, že leží v téže dílčí sítí (viz 44. díl našeho seriálu), a tak mu IP datagram pošle přímo. Tento případ je v terminologii TCP/IP označován jako tzv. **přímé směrování** (**direct routing**). Jeho režie je minimální: vzhledem k formátu IP adres (viz obr. 44.2. ve 44. dílu) je triviální rozpoznat, zda příjemce leží v téže dílčí síti, a pokud ano, dílčí síť umožňuje odeslat příslušný datagram jeho koncovému adresátovi přímo.

.PI OBR46_2.TIF, 20, 40, 10

Obr. 46.2.: Představa přímého směrování

Nyní si ale představme situaci na obrázku 46.3., kdy hostitelský počítač A chce odeslat IP datagram hostitelskému počítači C. Jelikož z jeho IP adresy rozpozná, že leží v jiné dílčí síti, pošle datagram bráně G (ve "své" dílčí síti). Tím úloha hostitelského počítače A jako odesilatele končí, a další je již na bráně G resp. na celé struktuře bran, které musí být schopné si datagram postupně předávat tak dlouho, dokud se nedostane do cílové dílčí sítě, kde pak může být prostřednictvím přímého směrování odeslán hostitelskému počítači, který je jeho konečným adresátem. Právě naznačený případ odpovídá tzv. **nepřímému směrování** (**indirect routing**), ke kterému na rozdíl od směrování přímého dochází tehdy, jestliže adresát není v téže dílčí síti jako odesilatel, a ten proto posílá datagram bráně.

.PI OBR46_3.TIF, 20, 40, 10

Obr. 46.3.: Představa nepřímého směrování

Volba mezi více bránami

Vzhledem k tomu, že vzájemně propojené sítě tvoří souvislý celek, musí být v každé dílčí síti vždy alespoň jedna brána. Nemusí ovšem být zdaleka jen jedna. Představme si jednoduchý příklad na obrázku 46.4., kde má hostitelský počítač A na výběr dvě brány, G1 a G2. Každá z nich by sice měla být schopna doručit IP datagram kamkoli je třeba, ale žádná z nich nebude zřejmě optimální pro všechny možné cíle - k některým vede kratší cesta přes bránu G1, k jiným zase přes bránu G2. Znalost toho, kdy je výhodnější poslat datagram bráně G1 a kdy bráně G2, by ale měl mít již hostitelský počítač A. Je samozřejmě možné, aby tento hostitelský počítač měl potřebné informace "pevně zabudovány" ve svých konfiguračních souborech, podle nichž si pak vytváří své směrovací tabulky. Ve vzájemně propojených sítích malého rozsahu a s minimem dynamických změn to je rozumné řešení, v případě větších a častěji se měnících konglomerátů sítí již nikoli. Zde již je nutný jiný mechanismus, umožňující provádět dynamickou aktualizaci směrovacích tabulek hostitelských počítačů podle okamžité situace.

Filosofie směrování v TCP/IP ale vychází z toho, že znát nejvhodnější cesty a průběžně reagovat na skutečnou situaci je úkolem bran, a nikoli hostitelských

počítačů. V soustavě protokolů TCP/IP je proto zabudován mechanismus, pomocí kterého je celý problém velmi elegantně řešen: kdykoli nějaká brána zjistí, že některý hostitelský počítač používá neoptimální cestu, upozorní jej na tuto skutečnost. Ukažme si to na příkladu (viz opět obrázek 46.4.): nechť hostitelský počítač A v dílčí síti X pošle bráně G1 IP datagram, určený počítači C v dílčí síti Y. Brána G1 zná optimální cestu a ví, že ze sítě X do sítě Y je výhodnější použít bránu G2. O doručení přijatého datagramu se brána G1 ještě postará, ale současně s tím upozorní hostitelský počítač A, že další datagramy, směřující do sítě Y, již má posílat přes bránu G2. Hostitelský počítač A si tuto informaci zanese do svých směrovacích tabulek, a nadále se podle ní řídí.

.PI OBR46_4.TIF, 20, 40, 10

Obr. 46.4.: Příklad dílčí sítě se dvěma bránami

Právě naznačený mechanismus má jeden velmi příjemný efekt - jednotlivé hostitelské počítače vystačí na počátku (tj. při svém spuštění) se znalostí jedné jediné brány ve "své" dílčí síti. Další brány se pak "naučí" používat díky právě popsanému mechanismu. Ten je v praxi zajišťován prostřednictvím protokolu **ICMP** (**Internet Control Message Protocol**), který je povinnou součástí protokolu IP, a slouží obecně pro předávání řídících informací a zpráv o chybách a nestandardních situacích.

.cp9

Připomeňme si však ještě jednu významnou skutečnost, kterou jsme si uvedli již ve 44. dílu našeho seriálu - kvůli minimalizaci rozsahu směrovacích tabulek je veškeré směrování v TCP/IP sítích založeno jen na dílčích sítích jako takových, a nikoli na jednotlivých hostitelských počítačích (přesněji: na té části IP adres, které představují adresy dílčích sítí). V našem konkrétním případě, odpovídajícím obrázku 46.4., to znamená, že hostitelský počítač A si ve svých směrovacích tabulkách bude pamatovat pouze to, že nejvýhodnější cesta do sítě Y vede přes bránu G2, a nikoli že k hostitelskému počítači C se dostane nejlépe přes bránu G2.

47/ Směrování v TCP/IP sítích - II.

V minulém díle jsme se začali podrobněji zabývat problematikou směrování v TCP/IP sítích. Řekli jsme si, že na směrování se podílí jak brány, tak i hostitelské počítače, a blíže jsme se zabývali právě rolí hostitelských počítačů. Dnes se podrobněji zastavíme u toho, jakým způsobem se na směrování podílí brány (IP směrovače).

Vraťme se však ještě jednou k hostitelským počítačům: jejich úloha při směrování končí tím, že datagram buď pošlou přímo jeho koncovému adresátovi (v případě tzv. přímého směrování, viz minule, kdy se příjemce nachází v téže dílčí síti), nebo tím, že datagram předají některé z bran (v opačném případě). O postupu hostitelského počítače v prvním případě jsme si již povídali v souvislosti s problematikou adresování a transformování IP adres (ve 45. dílu). Pro druhý případ musí mít hostitelský počítač právě tolik informací, aby dokázal zvolit jednu z bran ve své dílčí síti, a té datagram poslat. Tyto informace si hostitelský počítač uchovává ve svých směrovacích tabulkách, jejichž představu ilustruje obrázek 47.1.

.PI OBR47_1.TIF, 20, 40, 10

Obr. 47.1.: Představa obsahu směrovacích tabulek

Brána v roli učitele

V minulém dílu jsme si naznačili, že znát nejvhodnější cesty a tyto znalosti si průběžně aktualizovat je úkolem bran (IP směrovačů), a nikoli hostitelských počítačů. Ty mohou zpočátku posílat veškeré datagramy jedné jediné bráně ve své dílčí síti, a ta je vždy upozorní v případě, že by prostřednictvím jiné brány byla cesta datagramu výhodnější. Ukažme si nyní, jaký efekt má tento mechanismus na směrovací tabulky hostitelských počítačů (viz obr. 47.2.): po spuštění hostitelského počítače musí jeho směrovací tabulka obsahovat informace alespoň o jedné bráně. Tato brána (resp. jedna z bran, je-li ve směrovací tabulce uvedeno) je přitom prohlášena za implicitní (default), a hostitelský počítač jí z počátku posílá všechny datagramy, které směřují do jiné dílčí sítě. Jakmile tato implicitní brána zjistí, že by hostitelský počítač měl použít jinou výhodnější cestu (vedoucí přes jinou bránu), upozorní jej na to. Hostitelský počítač si na základě této explicitní informace upraví svou směrovací tabulku (viz obrázek 47.2) - zavede si v ní novou položku, a v ní si poznačí, že datagramy, směřující do příslušné dílčí sítě, má posílat přes tu a tu bránu. Kdykoli ale má odeslat datagram do dílčí sítě, pro kterou ještě nemá ve své směrovací tabulce uvedenu konkrétní cestu, pošle datagram té bráně, která je v jeho směrovací tabulce uvedena jako implicitní. Takováto organizace směrovacích tabulek velmi dobře odpovídá "stromovitým" konfiguracím (jako např. na obrázku 47.1.), které jsou u lokálních sítí dosti časté, a při kterých samostatné cesty (vyjádřené ve směrovací tabulce explicitně) vedou jen k několika málo dalším dílčím sítím, zatímco ke "zbytku světa" vede cesta jediná (zpřístupněná implicitní bránou).

.PI OBR47_2.TIF, 20, 40, 10

Obr. 47.2.: Představa průběžného doplňování směrovací tabulky hostitelského počítače

.cp9

Činnost bran

Ponechme ještě stranou otázku, jakým způsobem získávají jednotlivé brány potřebné informace o cestách (k tomu se dostaneme příště), a podívejme se, jak brány postupují při vlastním směrování.

Pravidla pro směrování datagramů se u bran principiálně neliší od hostitelských počítačů. Zjistí-li brána, že přijatý datagram je určen adresátovi v dílčí síti, do které je brána připojena (a takové musí být alespoň dvě, má-li jít vůbec o bránu), pošle mu datagram přímo - v rámci přímého směrování (viz obr. 47.3 a/). Je-li datagram určen adresátovi v jiné dílčí síti, do které není brána přímo připojena, musí zvolit jinou vhodnou bránu, a té datagram předat (viz obr. 47.3. b/). Postupuje přitom obdobně jako hostitelský počítač: pokud ve své směrovací tabulce najde explicitní údaj o tom, že pro danou cílovou síť má datagram předat té a té bráně, učiní tak. Pokud o dané cílové síti nemá ve své směrovací tabulce žádné údaje, použije tu bránu, kterou považuje za implicitní - pokud ovšem má ve své směrovací tabulce implicitní bránu

vůbec definovánu. Pokud ne, není daná brána schopna směrování datagramu zajistit, a musí to oznámit jako chybu.

.PI OBR47_3.TIF, 20, 40, 10

Obr. 47.3.: Směrování bránou

a/ přímé směrování

b/ nepřímé směrování

(další skok - next hop).

Obsah směrovacích tabulek

Vraťme se ještě jednou k obsahu směrovacích tabulek. Jak jsme si již několikrát naznačili, je z důvodu minimalizace jejich rozsahu veškeré směrování založeno jen na adrese cílové sítě, resp. na té části IP adresy, která tuto cílovou síť vyjadřuje. Směrovací tabulky jsou pak ve své podstatě vlastně seznamem dvojic <síť, brána>, kde "síť" představuje cílovou dílčí síť, a "brána" je ta z bran, přes kterou do cílové sítě vede (nejvhodnější) cesta. Jde přitom o bránu, která ještě není konečným příjemcem datagramu, ale jen jeho další přestupní stanicí na cestě k cíli. V angličtině se označuje jako **next hop**, doslova: další skok.

Bránou pro "další skok" však musí vždy být brána, která je z daného místa dosažitelná přímo - tedy taková brána , která se nalézá ve stejné dílčí síti, jako daný hostitelský počítač, resp. v některé z dílčích sítí, do kterých je přímo připojena daná brána.

Adresy bran, které jsou obsaženy ve směrovacích tabulkách, jsou zásadně IP adresy, ačkoli by se na první pohled mohlo zdát výhodnější používat zde přímo fyzické adresy - čímž by se ušetřila opakované transformace IP adresy na fyzickou adresu brány (viz 45. díl seriálu). Důvodem pro použití IP adres je možnost využívat ve všech uzlech stejnou programovou realizaci směrovacího mechanismu, nezávisle na konkrétních fyzických adresách. Dalším, velmi důležitým důvodem, je možnost průběžné aktualizace směrovacích tabulek a také možnost případného ladění, dojde-li k chybě či jiným nestandardním situacím při směrování. V neposlední řadě použití IP adres ve směrovacích tabulkách vychází i z celkové koncepce síťové (IP) vrstvy a protokolu IP - usilující o vytvoření jednotné abstrakce všech dílčích sítí.

Výjimka z pravidla

Nebylo by to snad ani žádné pravidlo, kdyby nemělo také své výjimky. V případě pravidla o směrování jen na základě adres dílčích sítí je touto výjimkou možnost zavést do směrovacích tabulek explicitně také údaje o cestách ke konkrétním hostitelským počítačům, nikoli jen do celých dílčích sítí. Ačkoli za běžného provozu by tato možnost měla být využívána spíše vyjímečně, je velmi užitečná pro správce sítí při ladění a správě sítě obecně. Zde může být k nezaplacení možnost přesměrovat tok služebních dat, směřujících k pracovní stanici správce tak, aby se vyhnul přetíženému či zcela neprůchodnému místu, které je zdrojem problémů.

48/ Směrování v TCP/IP sítích - III.

V minulých dvou dílech jsme se zabývali konkrétním postupem hostitelských počítačů a bran (IP směrovačů) při směrování IP datagramů. Ukázali jsme si, jakým způsobem k tomu oba druhy uzlů TCP/IP sítí využívají své směrovací tabulky, a naznačili jsme si také, že udržování správného obsahu směrovacích tabulek hostitelských počítačů mají vlastně na starosti brány. Dnes se již dostaneme k tomu, jak si své směrovací tabulky vytváří a udržují samotné brány.

Přístup hostitelských počítačů je založen na myšlence, že dokáží úspěšně směrovat datagramy i v případě, kdy mají k dispozici jen částečnou znalost nejvhodnějších cest v celém konglomerátu vzájemně propojených sítí (internetu). Ty dílčí sítě, do kterých jsou ve směrovací tabulce explicitně definovány cesty, odpovídají té části internetu, kterou vlastník směrovací tabulky "zná". Zbývající část internet-u je pak pro majitele směrovací tabulky reprezentována implicitní (default) bránou, resp. implicitní cestou, která přes tuto bránu vede.

Hlavní výhodou směrování s využitím implicitních cest je značná redukce rozsahu směrovacích tabulek, a s tím související menší režie na jejich aktualizaci a udržování konzistentního stavu. Nevýhodou je pak potenciální neefektivita směrování - cesta, vedoucí přes implicitní bránu, samozřejmě nemusí být nejkratší.

Ani brány nemusí znát všechno

Podobně jako v případě hostitelských počítačů, nebylo by ani v případě bran únosné, aby každá z nich "znala" vždy celý internet - tedy aby její směrovací tabulka obsahovala explicitní údaj o optimální cestě do každé jednotlivé dílčí sítě. Také zde se totiž brzy ukázalo, že již pro nepříliš velké soustavy vzájemně propojených sítí vychází "úplné" směrovací tabulky značně rozsáhlé. Jejich rozsah by však ještě nebyl tak velkým problémem, jakým se náhle stává udržování vzájemné konzistence a průběžná aktualizace směrovacích tabulek jednotlivých bran. K tomu je totiž nutný pravidelný přenos velmi velkého objemu dat, který může brzy zcela zahltit dostupné přenosové cesty.

Proto se i v případě bran aplikovala stejná myšlenka, jako u hostitelských počítačů: vybavit většinu z nich jen částečnou znalostí nejkratších cest do všech možných cílových sítí, a využívat implicitní směrování. Konkrétní realizace se pak vyvíjela na základě toho, jak se TCP/IP protokoly prosazovaly do nově vznikajícího Internetu. Podívejme se proto, jak se situace vyvíjela zde.

Na počátku byl ARPANET

V době, kdy v USA započaly první experimenty se sítí Internet, její zárodečná síť (ARPANET) již existovala a byla v provozu. Vůči právě vznikajícímu Internetu se pak tato rozlehlá síť chovala jako páteřní síť (backbone), ke které byly postupně připojovány jednotlivé lokální sítě či celé skupiny lokálních sítí, viz obr. 48.1.

.PI OBR48_1.TIF, 20, 40, 10

Obr. 48.1.: Představa vzniku Internetu

Pokud se jednalo jen o jednu lokální síť, byla tato napojena na páteřní síť prostřednictvím jediné brány. Pokud šlo o více lokálních sítí (například o skupinu lokálních sítí v rámci areálu univerzity apod.), byly tyto navzájem

propojeny dalšími bránami, ale na páteřní síť byly jako celek připojeny opět jen v jednom bodě, resp. prostřednictvím jediné brány (viz obrázek 48.2). Tato brána pak vůči ostatním vystupovala jako implicitní brána, přes kterou byl směrován veškerý provoz, určený "mimo" nově připojenou skupinu dílčích sítí.

Hlavní (core) a vedlejší (noncore) brány

Připojování celých soustav dílčích sítí na páteřní síť vždy jen přes jednu bránu dávalo vzniknout struktuře, ve které se s výhodou uplatnilo implicitní směrování, tedy směrování s využitím implicitní cesty. Všem bránám v určité skupině dílčích sítí stačilo k efektivnímu směrování znát optimální cesty jen v rámci dané skupiny sítí, a veškerý ostatní provoz směrovat přes implicitní bránu do páteřní sítě. Tyto brány tedy mohly pracovat jen s částečnou znalostí celého Internet-u, což značně redukovalo rozsah jejich směrovacích tabulek, a umožňovalo také určitou autonomii při zajišťování změn v "místním" směrování. Neefektivita, vznikající v důsledku implicitního směrování, se zde vzhledem ke skutečné topologii nemohla uplatnit.

.PI OBR48_2.TIF, 20, 40, 10

Obr. 48.2.: Představa hlavních a vedlejších bran při připojení skupiny dílčích sítí

Pro ty brány, které byly přímo připojeny na páteřní síť, však již stejná úvaha neplatila. V rámci páteřní sítě (která měla ve skutečnosti dosti složitou vnitřní strukturu) by při implicitním směrování již docházelo k výrazným neefektivitám. Tvůrci Internetu se proto rozhodli poskytnout všem bránám, připojeným přímo na páteřní síť, "úplnou" znalost celého Internetu, a umožnit jim tak využívat skutečně optimálních cest v rámci páteřní sítě. Každá z těchto tzv. hlavních bran **(**core gateways**) tedy znala optimální cestu do všech dostupných cílových sítí v celém Internetu. Všechny hlavní brány si pak navzájem pravidelně předávaly nezbytné směrovací informace, pomocí kterých si aktualizovaly a vzájemně koordinovaly své směrovací tabulky. Tvořily tak ucelený systém, který mohl fungovat dostatečně spolehlivě díky tomu, že všechny hlavní brány spravovala jediná centrální instituce (Internet Network Operations Center).**

Všechny ostatní brány, které nebyly přímo napojeny na páteřní síť (tzv. vedlejší brány**,** noncore gateways**), pak již spadaly plně do kompetence provozovatelů jednotlivých dílčích sítí či skupin dílčích sítí. Tito provozovatelé však měli ještě jednu důležitou povinnost - systému hlavních bran museli vhodně zveřejnit IP adresy všech svých dílčích sítí. Díky vzájemnému předávání směrovacích informací se pak již hlavní brány dokázaly samy podělit o znalost těchto dílčích sítí, které se tak záhy staly dostupné z celého rozsáhlého Internetu.**

Nic nevydrží věčně

Právě naznačené schéma se však záhy ukázalo být nepraktické, a to hned z několika důvodů. Jedním z nich byl neočekávaně velký růst Internetu. Jeho původní struktura, soustředěná kolem jediné páteřní sítě, se záhy stala značně složitou, a stejně tak se značně netriviálními staly i mechanismy udržování směrovacích tabulek hlavních bran ve vzájemně konzistentním stavu. Také režie

těchto mechanismů se při stále rostoucím počtu hlavních bran neúměrně zvyšovala. Kromě toho vznikly problémy i se samotnými hlavními bránami - zdaleka ne všechny lokality měly možnost zřídit pro své potřeby hlavní bránu, a připojit ji přímo na páteřní síť.

Situace si poměrně brzy vynutila řadu změn, ale o nich si povíme až příště.

49/ Směrování v TCP/IP sítích - IV.

V minulém dílu jsme dospěli k představě toho, že Internet vznikal a postupně se rozšiřoval připojováním jednotlivých dílčích sítí či celých skupin dílčích sítí přímo na zárodečnou páteřní síť. V rámci těchto skupin byly jednotlivé dílčí sítě propojeny tzv. vedlejšími bránami (noncore gateways), zatímco na páteřní síť byla celá skupina připojena jedinou, tzv. hlavní bránou (core gateway). Tato hlavní brána přitom pravidelně komunikovala se všemi ostatními hlavními bránami Internetu, a vyměňovala si s nimi potřebné směrovací informace. Díky tomu mohly všechny hlavní brány pracovat s úplnou znalostí celého Internetu, tj. znát nejkratší cestu do kterékoli dílčí sítě. Naproti tomu vedlejší (noncore) brány vystačily jen se znalostí svého bezprostředního okolí (resp. příslušné skupiny sítí, sdílejících stejnou hlavní bránu), a pro všechny ostatní cíle používaly implicitní směrování přes hlavní bránu.

Již minule jsme si ale také řekli, že vedlejší brány spadaly plně do kompetence provozovatelů příslušných skupin sítí, které tyto brány propojovaly (zatímco všechny hlavní brány spravovala jediná centrální autorita). Tito provozovatelé pak měli za úkol vhodně zveřejnit informace o dostupnosti svých dílčích sítí systému hlavních bran, který se o ně již dokázal sám podělit. Otázkou ovšem bylo, jak toto zveřejnění zajistit.

Pokud byla přes hlavní bránu připojena k páteřní síti jen jediná dílčí síť, byla situace triviální. Častěji ale byly k páteřní síti připojovány větší celky - celé skupiny dílčích sítí, vzájemně propojených vedlejšími bránami. Zde již byl zapotřebí vhodný mechanismus, který by kromě vlastního jednorázového zveřejnění umožnil také nezbytnou průběžnou aktualizaci směrovacích informací ve vztahu k hlavním bránám.

Problémem však nebyla ani tak technická stránka věci, jako spíše stránka organizační: která z vedlejších bran má předávat potřebné směrovací informace soustavě hlavních bran?

Kromě toho zde byl ještě jeden důležitý aspekt. Nově připojované skupiny vzájemně propojených sítí byly často samy o sobě rozsáhlé konglomeráty, které dříve fungovaly jako samostatné internet-y (s malým "i"), a měly tedy již vybudovány a implementovány vlastní mechanismy a prostředky řízení a správy. Jejich provozovatelé si samozřejmě chtěli i nadále udržet kontrolu nad svými sítěmi, ale současně s tím získat také všechny výhody, plynoucí z přímého napojení na Internet.

Řešením pak bylo přijetí koncepce tzv. **autonomních systémů** (**autonomous systems**). Jejich myšlenka je velmi jednouchá: vzájemně propojené sítě, které spadají pod společnou správu, budou tvořit jediný autonomní systém, za který bude plně odpovídat jeho provozovatel. Současně s tím však bude existovat jednotný způsob vzájemného předávání směrovacích informací mezi jednotlivými autonomními systémy, který budou všichni povinni dodržovat. Jinými slovy: v rámci svého

vlastního systému má každý možnost zajistit si přenos a aktualizaci směrovacích údajů podle svého, ale navenek musí všichni postupovat jednotně.

Aby toto řešení bylo důsledné, stala se autonomním systémem také původní páteřní síť, resp. celá soustava hlavních bran. Tím se celá struktura Internetu logicky zjednodušila na jediný strom, jehož kořenem je právě autonomní systém, tvořený soustavou hlavních bran - viz obrázek 49.1, srovnej s obr. 48.1. a 48.2.

.PI OBR49_1.TIF, 20, 40, 10

Obr. 49.1.: Představa autonomních systémů

Pro vzájemnou komunikaci mezi autonomními systémy pak byl navržen potřebný protokol **EGP** (**Exterior Gateway Protocol**). Ten vychází z představy, že v rámci každého autonomního systému je jedna brána pověřena předáváním směrovacích informací navenek. Tato brána, jejíž výběr je plně v moci správce příslušného autonomního systému, je pak označována jako **externí brána** (**exterior gateway**). Prostřednictvím protokolu EGP pak tato brána komunikuje s externími bránami jiných autonomních systémů - viz obrázek 49.2.

.PI OBR49_2.TIF, 20, 40, 10

Obr. 49.2.: Působnost protokolů EGP a IGP

Protokol EGP umožňuje, aby se jeden autonomní systém nejprve dohodl s jiným, že si směrovací informace vůbec budou vyměňovat (nebo si je také mohou odmítnout předávat). Dále umožňuje, aby jednotlivé autonomní systémy pravidelně testovaly dostupnost druhých autonomních systémů (tj. testovaly, zda spojení s nimi je průchodné). Hlavním úkolem protokolu EGP však je pravidelný přenos směrovací údajů a informací o průběžných změnách.

Protokol EGP je tedy prostředkem, pomocí kterého může autonomní systém informovat jiné autonomní systémy o dostupnosti dílčích sítí v rámci vlastního autonomního systému, a naopak získávat informace o cestách do dílčích sítí v jiných autonomních systémech. V Internetu ale vzhledem k jeho konkrétní stromovité topologii stačí, když jednotlivé autonomní systémy předávají potřebné směrovací informace soustavě hlavních bran (která je sama o sobě autonomním systémem).

Každý autonomní systém však smí zveřejňovat údaje o dostupnosti pouze pro ty dílčí sítě, které spadají pod jeho správu, tedy jen pro dílčí sítě v rámci vlastního autonomního systému. Nesmí fungovat jako prostředník, který by sám zveřejňoval údaje, získané od jiného autonomního systému.

Jednou ze zajímavých vlastností protokolu EGP je skutečnost, že sice přenáší údaje o "délkách" cest do jednotlivých dílčích sítí, ale pak je již nepoužívá. V důsledku toho vlastně pracuje jen informaci o tom, zda cesta do příslušné dílčí sítě existuje či nikoli. To má zajímavý důsledek: kdyby do některé dílčí sítě existovalo více možných cest, příslušné brány by neměly podle čeho mezi nimi vybírat. Proto jsou při vzájemném propojování autonomních systémů zakázány takové topologie, které by připouštěly

více možných cest do stejných cílových sítí. Protokol EGP tedy počítá pouze se stromovitou strukturu, což ale v případě Internetu plně postačuje.

Pokud jde o přenos směrovacích informací v rámci jednotlivých autonomních systémů, zde mají jejich provozovatelé možnost vlastního výběru. Mezi možnosti, které se jim nabízí, patří také samotný protokol EGP, který lze používat i "uvnitř" autonomních systémů (povinný je pouze "vně", mezi autonomními systémy). Další, dosti rozšířenou možností je protokol **RIP** (**Routing Information Protocol**), prostřednictvím kterého si jednotlivé vnitřní brány autonomního systému předávají údaje o tom, do kterých cílových sítí vede z které brány cesta a jak je dlouhá (měřeno v počtu mezilehlých sítí). Každá z bran si pak na základě těchto údajů sama dopočítává nejkratší cesty do všech dostupných dílčích sítí způsobem, který jsme si naznačili již v 34. dílu našeho seriálu. Na podobném principu je založen alternativní protokol **Hello**, který však měří délky jednotlivých cest nikoli v počtu mezilehlých sítí, ale v délce zpoždění při přenosu po této cestě. Společnou nevýhodou těchto algoritmů, které pracují s délkou cest (a jsou proto označovány jako **vector-distance algorithms**) je jejich velká režie, která navíc při zvětšování počtu bran velmi rychle roste. Z tohoto pohledu jsou pak výhodnější algoritmy, označované společně jako **link-state** či **SPF** (**Shortest Path First**) algoritmy. Ty předpokládají, že každá brána zná topologii celého konglomerátu vzájemně propojených sítí, a pravidelně testuje průchodnost přenosových cest ke svým bezprostředním sousedům. Dvouhodnotové informace o této dostupnosti pak rozesílá všem ostatním bránám, které si na jejich základě, podle tzv. Dijkstrova algoritmu, samy vyhodnocují nejkratší cesty do všech dílčích sítí. Ve srovnání s předchozí skupinou vykazují tyto algoritmy výrazně nižší režii, která při zvětšování počtu bran navíc neroste tak rychle.

Zajímavý je z tohoto pohledu také způsob, jakým si potřebné směrovací informace vyměňovala soustava hlavních bran, nyní tvořící jeden autonomní systém. Původně tyto hlavní brány používaly protokol **GGP** (**Gateway-to-Gateway Protocol**), který patří do stejné skupiny, jako protokoly RIP a Hello, tedy mezi protokoly, přenášející údaje o délce jednotlivých cest (v případě protokolu GGP měřené v počtu "přestupních" bran). Vzhledem k velké režii (viz závěr minulého dílu) však byl tento protokol nahrazen protokolem SPREAD (patřící mezi protokoly SPF), na který soustava hlavních bran přešla v roce 1988.

Všechny protokoly, používané pro přenos směrovacích informací mezi jednotlivými bránami "uvnitř" autonomních systémů, se souhrnně označují jako protokoly **IGP** (**Interior Gateway Protocol**) - viz též obrázek 49.2.

50/ Směrování v TCP/IP sítích - V.

V minulých dílech našeho seriálu jsme se podrobněji zabývali otázkou směrování v TCP/IP sítích. Konkrétní mechanismy, které jsme si v této souvislosti popisovali, však vycházely z určité strategie pro přidělování IP adres, kterou jsme si ukázali již ve 44. dílu. Tato strategie se ale s vývojem Internetu také postupně vyvíjela, a tak muselo nutně dojít k určitým modifikacím i ve způsobu směrování.

O změně strategie při přidělování IP adres jsme se již zmínili ve 44. dílu, kde jsme si o IP adresách povídali podrobněji. Připomeňme si proto, v čem tyto změny spočívají, a hlavně čím jsou motivovány.

Původní záměr byl takový, aby každá dílčí síť měla vlastní samostatnou síťovou adresu. Aby to bylo možné v prostředí, kde se dílčí sítě mohou velmi výrazně lišit co do počtu svých koncových uzlů (hostitelských počítačů, hosts), zavedly se tři různé formáty IP adres - adresy třídy A, B a C (viz 44. díl). Adresy třídy A byly určeny pro sítě s velmi velkým počtem hostitelských počítačů, adresy třídy B pro "středně" velké sítě, a adresy třídy C pro sítě s maximálně 254 hostitelskými počítači. Dlužno ovšem podotknout, že tato strategie vznikla ještě v době, kdy světu vládly velké střediskové počítače a minipočítače, zatímco éra osobních počítačů a malých lokálních sítí měla nastat přibližně o deset let později.

Takový růst nečekali

Tvůrci síťového modelu TCP/IP jistě předpokládali nárůst v nasazení výpočetní techniky, ale s čím již počítali méně je skutečnost, že tento nárůst bude tak prudký, a že se bude ubírat cestou "malých" počítačů a velkého množství relativně malých lokálních sítí. Zatímco zásoba IP adres třídy C ještě zdaleka vyčerpána není, úzkým místem se ukázala být jednak agenda, spojená s přidělováním IP adres, a především rozsah směrovacích tabulek, s nímž pak souvisí i vysoká režie na udržování jejich průběžné konzistence.

Jak jsme si již několikrát ukázali, snaží se TCP/IP protokoly minimalizovat rozsah směrovacích tabulek a režii na jejich aktualizaci dvěma způsoby: směrováním jen na základě adresy dílčí sítě, a používáním implicitního směrování (resp. implicitních cest) v tzv. vedlejších bránách (noncore gateways). Ani to však nestačí, a tak se ukázalo jako nezbytné usilovat o redukci počtu různých IP adres celých sítí. Jinými slovy: usilovat o tom, aby více dílčích sítí mělo přidělenu a sdílelo stejnou síťovou adresu (přesněji: tu část IP adresy, která představuje adresu sítě), a ve směrovacích tabulkách tak zabíralo jen jednu položku. Přišlo se samozřejmě na více možností, jak toho dosáhnout - tou nejefektivnější se ukázala být technika tzv. **podsítí** (**subnets**), kterou jsme si naznačili již ve 44. dílu našeho seriálu, a která je dnes již povinnou součástí TCP/IP protokolů. Zopakujme si proto její základní myšlenku.

Podsítě a jejich adresy

Určitá skupina dílčích sítí, které by bez použití techniky podsítí měly samostatné adresy, má naopak jednu společnou adresu, a vůči svému okolí vystupuje jako jediný celek, tj. jako jediná dílčí síť (viz obr. 50.1.).

.PI OBR50_1.TIF, 20, 40, 10

Obr. 50.1.: Představa podsítí

V rámci příslušné skupiny sítí je ale jejich společná IP adresa dále členěna - přesněji ta její část, která navenek představuje číslo (adresu) hostitelského počítače, se nyní rozpadá na dvě části - číslo (adresu) dílčí sítě, resp. tzv. **podsítě** (**subnet**) v rámci skupiny, a na číslo (adresu) hostitelského počítače v rámci této podsítě - viz obrázek

50.2. O tom, jak konkrétně má být IP adresa takto členěna, rozhoduje tzv. **maska podsítě** (**subnet mask**).

Podstatná je přitom skutečnost, že tato maska nemá žádný povinný tvar. Volí se samostatně pro každou jednotlivou podsíť, což pak ale znamená, že i v rámci jedné skupiny podsítí mohou být používány různé masky. Příslušný standard to sice umožňuje, ale nedoporučuje, neboť tak mohou snadno vznikat nejednoznačnosti v IP adresách. Obvykle se tedy setkáme spíše s tím, že v rámci jedné skupiny podsítí je používána stejná maska.

.PI OBR50_2.TIF, 20, 40, 10

Obr. 50.2.: Maska podsítě a její význam

Směrování v případě podsítí

Jakmile je ale používána technika podsítí, je nutné poněkud upravit konkrétní algoritmus směrování, který jsme si popsali v 47. dílu našeho seriálu. Doposud jsme totiž předpokládali, že ve směrovacích tabulkách jsou dvojice "cílová síť, brána", popisující jednotlivé dílčí sítě, a že algoritmus směrování v nich hledá na základě adresy cílové sítě v příslušném datagramu. Nyní je ale zapotřebí, aby ve směrovacích tabulkách byly trojice: "maska podsítě, cílová síť, brána", popisující i jednotlivé podsítě, a modifikovaný algoritmus směrování musí v takovýchto tabulkách hledat opět podle adresy cílové sítě, s tentokrát navíc s uvážením konkrétní masky podsítě pro každou jednotlivou položku zvlášť.

Tento modifikovaný algoritmus má jednu výhodu - dokáže zobecnit i ošetřit singularity, které se vyskytují u původního algoritmu (viz opět 47. díl), a to směrování na základě adres jednotlivých hostitelských počítačů, a použití implicitních cest. V prvním případě stačí volit takovou masku, díky které bude jako adresa sítě interpretována celá IP adresa příslušného hostitelského počítače (zatímco část, představující číslo hostitelského počítače, bude prázdná). V případě implicitních cest pak stačí použít opačný extrém - masku, která zcela "zamaskuje" adresu sítě.

Otázkou ovšem je, kdo má takto modifikovaný algoritmus směrování používat. Kdyby jej totiž musely používat všechny brány, zcela by se tím negoval výsledný efekt použití techniky podsítí, spočívající v další redukci směrovacích tabulek. Modifikovaný algoritmus směrování musí v každém případě používat všechny brány a hostitelské počítače v jednotlivých podsítích. Pokud jde o ostatní brány a hostitelské počítače, zde platí jednoduché teoretické pravidlo: modifikovaný algoritmus směrování je třeba použít tam, odkud mohou vést různé optimální cesty do různých podsítí se stejnou síťovou adresou.

Toto pravidlo lze jednoduše aplikovat v případě, že do celé skupiny podsítí se stejnou adresou vede jen jediný vstupní bod (jako například na obrázku 50.1.). V ostatních případech je situace mnohem komplikovanější.

Proxy ARP

Jak jsme si již naznačili výše, technika podsítí (subnetting) není jediným způsobem, jak může více dílčích sítí sdílet tutéž adresu. Další možností je mechanismus, označovaný jako **proxy ARP** či **promiscuous ARP**, nebo také **ARP hack**. Ten je ovšem použitelný jen tam, kde je využíván protokol ARP pro transformaci IP adres na adresy fyzické (tj. například v lokálních sítích typu Ethernet, viz 45. díl seriálu). Myšlenka proxy ARP je založená na jednoduchém triku, který ilustruje obrázek 50.3.:

.PI OBR50_3.TIF, 20, 40, 10

Obr. 50.3.: Představa Proxy ARP

a/ při reakci na žádost o fyzickou adresu

b/ při vlastním přenosu dat

brána G se snaží předstírat síti A, že je tím hostitelským počítačem (v síti B), se kterým chce právě komunikovat některý z uzlů sítě A. Dělá to tak, že kdykoli je pomocí protokolu ARP vyslán v síti A dotaz na fyzickou adresu některého hostitelského počítače v síti B, brána G na tento dotaz odpoví, a vrátí svou fyzickou adresu. Všechny datagramy, určené příslušnému počítači, jsou pak posílány na tuto fyzickou adresu, tedy ve skutečnosti bráně G, které je pak prostřednictvím sítě B dále předává jejich konečnému adresátovi. Výsledný efekt je tedy takový, jako kdyby síť B splývala se sítí A, a měla s ní tudíž stejnou síťovou adresu.

Problém je ovšem v omezené použitelnosti (jen tam, kde je používán protokol ARP), a dále také v tom, že některé implementace protokolu ARP se snaží detekovat a hlásit správci sítě tzv. **spoofing**, jak se označuje situace, kdy jedno zařízení předstírá, že je jiným zařízením (například proto, aby mohlo neoprávněně "odposlouchávat" přenášená data).

51. Jména v TCP/IP sítích - I.

V minulých dílech našeho seriálu jsme si ukázali, že v sítích na bázi protokolů TCP/IP (a samozřejmě také v Internetu) se používají jednotné IP adresy číselného charakteru, a že také veškeré směrování je založeno právě na těchto adresách. Jakkoli jsou ale tyto číselné adresy vhodné pro příslušný síťový software, pro uživatele příliš výstižné nejsou. Lidé raději pracují se snáze zapamatovatelnými symbolickými jmény, například "aviion.mff.cuni.cs", než s málo mnemonickými číselnými adresami typu 129.1.2.3. Jak je to ale s možností používat takováto symbolická jména v TCP/IP sítích?

Zamysleme se nejprve znovu nad tím, co je vlastně IP adresa. Ve 44. dílu jsme dospěli k tomu, že IP adresa je dvousložková, a že její dvě složky reprezentují jednak adresu dílčí sítě, jednak adresu hostitelského počítače (resp. brány) v této dílčí síti. Hostitelské počítače však mohou být přímo připojeny i do dvou či více dílčích sítí (pak jsou v angličtině označovány jako **multihomed hosts**), a brány dokonce musí být takto připojeny nejméně do dvou sítí, mají-li vůbec fungovat jako brány. Pak ovšem mají také dvě či více různých IP adres. Domyslíme-li tuto skutečnost do důsledku, je přesnější chápat IP adresu nikoli jako adresu počítače resp. brány jako takové, ale jako rozhraní mezi sítí a hostitelským počítačem resp. bránou. Navíc, dojde-li k přemístění počítače či brány, jejich IP adresy se změní.

Jména vs. IP adresy

Symbolická jména bran a hostitelských počítačů, která skutečně lze v TCP/IP sítích používat, však nejsou přesným symbolickým ekvivalentem číselných IP adres. Jména se vztahují k počítačům resp. bránám jako takovým, a nikoli k jejich rozhraním. Jeden hostitelský počítač tedy může mít jediné jméno, ale současně více IP adres (je ovšem také možné, aby měl jmen několik). Dojde-li k přitom k přemístění počítače, jeho jméno (či jména) se měnit nemusí (zatímco jeho IP adresa resp. adresy ano).

Otázkou ovšem je, jaký tvar by měla symbolická jména mít, kdo je oprávněn je přidělovat, a jak z těchto jmen získávat jim odpovídající IP adresy, které jsou potřeba pro vlastní komunikaci.

Flat namespace

Je-li třeba přidělit symbolická jména relativně malému počtu hostitelských počítačů a bran, je vše jednoduché - stačí vymyslet dostatečný počet navzájem různých jmen, a ty jednotlivým počítačům a bránám přidělit. Na syntaktický tvar těchto jmen přitom není třeba klást nějaké zvláštní požadavky - samozřejmě kromě jejich jednoznačnosti - a není ani nutné tato jména jakkoli strukturovat či členit na části. Všechna takováto symbolická jména jsou pak považována za dále nedělitelná (v angličtině se říká, že tvoří "**flat namespace**", doslova: plochý prostor jmen). Udržet takovýto systém dále nedělitelných jmen v konzistentním stavu však vyžaduje existenci centrální autority, která bude dbát na přípustnost nově přidělovaných jmen - hlavně podle toho, jestli nekolidují s některým již přiděleným jménem. Se zvyšováním počtu přidělovaných symbolických jmen však začne narůstat administrativní i technická zátěž této centrální instituce, a problematické se stává i hledání nových, dosud nepoužitých symbolických jmen. V takových kolosech, jako je Internet, se právě naznačené řešení stalo velmi brzy neúnosné.

Hierarchická jména

Je ale vůbec možné obejít se bez centrální instituce, která by dbala na konzistentnost celého systému symbolických jmen? Odpověď je kladná, a spočívá v tom, že právo přidělovat nová jména bude decentralizováno, a s ním i související odpovědnost za jejich jednoznačnost.

Způsob této decentralizace byl inspirován velkými organizacemi, které jsou hierarchicky členěny na různé organizační složky, a ve kterých vyšší složky delegují některé pravomoci složkám nižším. Podobně jsou na více složek členěna i symbolická jména, a tyto jejich složky jsou hierarchicky uspořádány - od nejvyšší až po nejnižší. Za nejvyšší složku stále zodpovídá jediná centrální autorita, ale za složky nižší úrovně již mohou odpovídat jiné subjekty, kterým centrální autorita deleguje příslušnou pravomoc.

Ukažme si to na hypotetickém příkladu sítě, propojující vysoké školy v Praze, a na příkladu jmen o třech složkách, které vyjadřují vysokou školu jako takovou, fakultu v rámci této vysoké školy, a jméno příslušného počítače na této fakultě. Adresy tohoto typu by mohly vypadat například takto:

aviion.mff.cuni

(počítač se jménem aviion na Matematicko-fyzikální fakultě (mff) Univerzity Karlovy (cuni)), nebo:

vax.fel.cvut

(tj. počítač se jménem vax na Elektrotechnické fakultě pražského ČVUT).

.cp9

Jednotlivé složky těchto hierarchických jmen jsou přitom zapisovány tak, že složka nejvyšší úrovně je nejvíce vpravo, a jednotlivé složky jsou oddělovány tečkami.

Pro tento námi zvolený příklad by stále ještě musela existovat centrální autorita, která by však odpovídala jen za jména, resp. složky nejvyšší úrovně, vyjadřující jména jednotlivých vysokých škol (tj. **cvut, cuni** atd.). Každá nová vysoká škola, která by se chtěla zapojit do tohoto systému jmen, by se musela obrátit na tuto centrální autoritu pro přidělení nového jména (resp. jeho nejvyšší složky). Centrální autorita pak přidělením každého nového jména nejvyšší úrovně vlastně vytváří samostatný prostor symbolických jmen, kterému se obvykle říká **doména** (v našem případě jde např. o domény **cvut**, **cuni**). Tím však odpovědnost centrální autority končí, neboť správu každé domény - tj. odpovědnost za přidělování dalších složek jmen - deleguje jednotlivým vysokým školám, které se stávají správci příslušných domén. Ti si pak sami volí jména (složky) druhé úrovně, v našem případě odpovídající jednotlivým fakultám (tj. například **mff, fel**). Tím vznikají další domény nižší úrovně (subdomény), které opět mají své správce - v našem případě jednotlivé fakulty - které si zase samy volí jména třetí úrovně. A jelikož jsme se v našem hypotetickém příkladu omezili jen na tři úrovně, jde již o jména konkrétních počítačů na příslušných fakultách. V praxi ovšem není počet úrovní (resp. složek) omezen.

Systém doménových jmen

Mechanismus, který v TCP/IP sítích implementuje právě naznačený systém hierarchických jmen, se jmenuje **DNS** (**Domain Name System**). Ten má dvě části - první z nich předepisuje syntaxi hierarchických jmen (označovaných nyní jako **doménová jména**), a pravidla pro delegování pravomocí a odpovědnosti za jejich přidělování. Druhá část pak určuje způsob implementace distribuovaného systému, který umožňuje efektivně převádět doménová jména na jim odpovídající IP adresy (o tom si budeme povídat příště).

Všechno podle Internetu

Standard DNS nepředepisuje žádný povinný tvar jednotlivých složek doménových jmen. Nepředepisuje ani, že členění na domény různých úrovní musí odpovídat organizačnímu členění - stejně tak dobře může toto členění vycházet například z územního členění.

Každý konglomerát vzájemně propojených sítí (internet) na bázi TCP/IP si tedy může volit způsob členění na jednotlivé domény i jejich jména zcela podle svého (viz náš hypotetický příklad). V praxi se ale téměř všechny takovéto konglomeráty přizpůsobují tomu, jak jsou doménová jména organizována v Internetu - mj. i z toho prostého důvodu, aby na něj mohly být co nejsnáze připojeny (pokud ještě nejsou).

Systém doménových jmen, používaný v "síti" Internet, má jeden pikantní rys - vznikl totiž v době, kdy jeho autoři vůbec neuvažovali o tom, že by se vůbec někdy rozšířil mimo území USA. Proto zavedli domény nejvyšší úrovně (tzv. **top-level domains**, viz tabulka 51.1.), které vyjadřují charakter příslušných organizací, ale jsou použitelné jen pro takové organizace, které sídlí v USA (a v části Kanady). Jakmile se ale Internet dostal i mimo Spojených států, musely se zavést ještě další domény nejvyšší úrovně, odpovídající jednotlivým státům. Jména těchto domén nejvyšší úrovně tvoří dvoupísmenové národní kódy (Country Code), které definuje standard ISO 3166, pro Československo pak: **cs**. Takže například

xinu.cs.purdue.edu

je počítač (xinu) na katedře computer science (cs) na Purdue University (purdue), která je vzdělávací institucí (edu), v USA. Naopak

aviion.mff.cuni.cs

je počítač (aviion) na Matematicko-fyzikální fakultě (mff) Univerzity Karlovy (cuni) v Československu (cs).

Náš hypotetický příklad tedy nebyl tak daleko od skutečnosti. Pouze jsme v něm neuvažovali, že celé Československo tvoří v rámci Internetu jednu doménu nejvyšší úrovně (doménu **cs**), jejímž správcem je pražská VŠCHT. U ní pak musí být registrovány všechny domény druhé úrovně.

doména	význam
com	výdělečné organizace
edu	vzdělávací instituce (školy)
gov	vládní instituce
mil	vojenské instituce
net	provozní a správní střediska sítí
org	ostatní organizace
int	mezinárodní organizace
národní kód	stát

Tabulka 51.1.: Domény nejvyšší úrovně v síti Internet

52/ Jména v TCP/IP sítích - II.

V minulém dílu jsme se zabývali otázkou symbolických jmen počítačů a bran v TCP/IP sítích a dospěli jsme k tomu, jakým způsobem se přidělují doménová jména v Internetu. Nyní se budeme věnovat otázce, jak na základě symbolických jmen získávat odpovídající IP adresy.

Připomeňme si ještě jednou, že otázku doménových jmen v TCP/IP sítích řeší standard DNS (Domain Name System), který má dvě základní části: první z nich určuje syntaxi doménových jmen a pravidla pro delegování pravomoci a odpovědnosti za jejich přidělování, zatímco druhá část se týká implementace mechanismu pro převod doménových jmen na jim odpovídající IP adresy.

Jestliže pravomoc a odpovědnost za přidělování doménových jmen je distribuována na jednotlivé domény a subdomény, je vcelku přirozené očekávat, že také mechanismus pro převod těchto doménových jmen na IP adresy bude obdobně distribuován - že bude tvořen soustavou vzájemně spolupracujících částí, nazývaných **servery jmen** (**name servers**), jejichž uspořádání bude odrážet hierarchickou strukturu domén a subdomén. Základní myšlenka takovéhotořešení je založena na tom, že s každou doménou a subdoménou bude spojen prostředek (zmíněný server jmen), který všechna jména z příslušné domény či subdomény buď umí převést sám, nebo alespoň zná jiný server jmen, který to dokáže.

.PI OBR52_1.TIF, 20, 40, 10

Obr. 52.1.: Část adresového prostoru doménových jmen

Představme si příklad na obrázku 52.1., který ukazuje malou část adresového prostoru doménových jmen Internetu se třemi doménami nejvyšší úrovně - **edu** (pro vzdělávací instituce v USA), **com** (pro komerční instituce v USA), a **cs** (pro celé Československo), s některými jejich subdoménami. Dále si představme stejně strukturovaný systém serverů jmen pro jednotlivé domény a subdomény dle obrázku 52.2. a předpokládejme, že některý hostitelský počítač potřebuje odeslat zprávu počítači s adresou **aviion.mff.cuni.cs**. Obrátí se proto nejprve na server jmen domény cs, který jej odkáže na server jmen domény cuni.cs, a ten pak zase na server jmen domény **mff.cuni.cs**. Ten již je schopen odpovědět na dotaz, jaká je IP adresa počítače **aviion.mff.cuni.cs.**

.PI OBR52_2.TIF, 20, 40, 10

Obr. 52.2.: První představa struktury serverů jmen

Jak je tomu ve skutečnosti

Právě naznačený příklad je samozřejmě dosti zjednodušený, ale jinak dobře vystihuje celkovou filosofii převodu symbolických doménových jmen na IP adresy.

Prvním praktický problém vyvstává hned v okamžiku, kdy je třeba se obrátit na server jmen domény nejvyšší úrovně - každý potenciální tazatel by totiž musel znát

adresy serverů jmen všech domén nejvyšší úrovně. Proto se všem těmto serverům nejvyšší úrovně nadřazuje ještě jeden, tzv. **kořenový server** (**root server**), a pouze tento kořenový server musí být znám všem potenciálním žadatelům o převod doménových adres. V praxi je tento kořenový server několikanásobně zálohován.

Další odlišností skutečné realizace od naší ideální představy je hloubka výsledného stromu serverů jmen, v jehož kořeni je právě zmíněný kořenový server. Jednotlivé servery jmen totiž mohou obsahovat potřebné informace o doménových jménech pro více různých domén a subdomén - velice často totiž různé organizace soustřeďují informace o jménech ze všech svých subdomén v jediném serveru jmen. Kořenový server naopak obsahuje informace i o všech doménách nejvyšších úrovních, takže výsledný strom serverů jmen bývá v praxi mnohem "mělčí" (viz obr. 52.3.), než naše původní představa na obrázku 52.2.

.PI OBR52_3.TIF, 20, 40, 10

Obr. 52.3.: Realističtější představa struktury serverů jmen

Rekurzivní a iterativní převod

Iniciátorem převodu doménového jména na IP adresu (anglicky: **name resolution**) je vždy programová entita hostitelského počítače (tzv. **name resolver**), která vůči celé soustavě serverů jmen vystupuje jako klient. Se svým požadavkem na převod doménového jména se tato entita obrací na některý ze serverů, který může postupovat dvojím způsobem: pokud není schopen převod zajistit, sám se obrátí na jiný server jmen, který převod zajistí, výsledek vrátí prvním serveru, a ten jej pak vrátí původnímu žadateli. Pak jde o tzv. **rekurzivní převod** (**recursive resolution**). Alternativou je tzv. **iterativní převod** (**iterative resolution**), při kterém dotázaný server jmen buď provede převod sám, nebo pouze vrátí adresu jiného serveru jmen, na který se pak musí žadatel o převod znovu obrátit sám.

Zajímavou otázkou je to, kam se má žadatel o převod obrátit nejprve - zda má začít shora, a obrátit se na kořenový server, nebo má naopak postupovat naopak odspodu, a obrátit se nejprve na "místní" server jmen? Ve prospěch druhé možnosti hovoří hned několik skutečností: kdyby se všichni obraceli přímo na kořenový server, tento by byl brzy zahlcen. Navíc požadavky na převod doménových jmen se nejčastěji týkají právě "místních" jmen, které dokáže převést místní server. Ten je kromě toho schopen fungovat i v případě eventuálního výpadku vyšších vrstev celého stromu name serverů (i když to je vzhledem k jejich zálohování nepříliš pravděpodobné).

Použití vyrovnávacích pamětí (caching)

Kdyby se každý hostitelský počítač musel obracet na soustavu serverů jmen pokaždé, kdy potřebuje převést některé doménové jméno na jemu odpovídající IP adresu, byla by to pro celou síť neúnosně velká zátěž. Pro zefektivnění celého mechanismu převodu se proto počítá s tím, že hostitelské počítače si po určitou dobu pamatují výsledky dříve uskutečněných převodů. Každý hostitelský počítač si proto bude udržovat ve vhodné vyrovnávací paměti (paměti cache) databázi symbolických jmen a jim odpovídajících IP adres. Aby tuto svou databázi udržel v konzistentním stavu, odpovídajícím skutečnosti, bude každá položka této databáze "zastarávat" - po určité

době ztratí svou platnost, a příslušný převod bude muset být v okamžiku potřeby vyvolán znovu.

Podobně postupují i servery jmen. Kromě doménových jmen, které jsou schopny (přesněji: oprávněny) převádět samy, se při rekurzivních převodech dozvídají i odpovídající IP adresy k jiným doménovým jménům. Také ty si udržují ve své vyrovnávací paměti, a v případě žádosti o převod je mohou poskytnout - ovšem s poznámkou, že nejsou k jejich převodu kompetentní (tj. že jde o tzv. **neautoritativní** (**nonauthoritative**) převod). Současně s tím poskytnou žadateli o převod i odkaz na ten server jmen, který je k převodu kompetentní. Iniciátor převodu pak naloží s touto informací podle vlastního uvážení. Jde-li mu o rychlost, použije tento neautoritativní převod, jde-li mu naopak o spolehlivost, obrátí se na příslušný server jmen, který je pro daný převod kompetentní.

Ještě většího zefektivnění celého mechanismu převodu doménových jmen na IP adresy lze pak dosáhnout tím, že jednotlivé hostitelské počítače si v okamžiku svého spuštění vyžádají od místního serveru jmen celou jeho databázi doménových jmen a jim odpovídajících IP adres. Z té pak vychází, a na server jmen se obrací jen při její aktualizaci (po "zastarání" některé položky) či pro její doplnění o nové doménové jméno, jehož odpovídající IP adresa ještě v databázi není.

53/ Protokol IP

V předcházejících dílech našeho seriálu jsme se zabývali adresami a způsobem adresování na různých úrovních - tedy fyzickými adresami, IP adresami i symbolickými jmény, a dále otázkou směrování. Nyní se již budeme věnovat konkrétním mechanismům přenosů v TCP/IP sítích.

Vraťme se však ještě jednou ke 42. dílu, ve kterém jsme si naznačili celkovou filosofii síťového modelu TCP/IP, a srovnávali ji s filosofií referenčního modelu ISO/OSI. Zde jsme si uvedli, že síťový model TCP/IP vychází z předpokladu, že zajištění spolehlivosti při přenosu dat je spíše otázkou koncových účastníků, a nikoli otázkou komunikační podsítě. Řečeno jinými slovy: síťový model TCP/IP zařazuje mechanismy pro zajištění spolehlivosti přenosů až do vrstvy transportní, a nikoli do vrstvy síťové (resp. tzv. IP vrstvy). Od této vrstvy očekává pouze jednoduchou přenosovou službu nespojovaného (connectionless) charakteru, která by sice neměla samovolně ztrácet přenášené pakety, ale na druhé straně zase není povinna podnikat žádné nápravné akce v případě, že se některý paket skutečně ztratí, nebo když jej tato vrstva není z objektivních důvodů schopna doručit - například kvůli přeplněným vyrovnávacím pamětem, kvůli výpadku spojení atd. Takovouto nespolehlivou nespojovanou službu však musí síťová vrstva poskytovat jednotným způsobem v rámci celé soustavy vzájemně propojených sítí (tj. v rámci celého internetu, s malým "i"), bez ohledu na to, zda jsou data ve skutečnosti přenášena po lokální síti typu Ethernet, po dvoubodovém spoji, veřejnou datovou sítí či jiným způsobem.

.PI OBR53_1.TIF, 20, 40, 10

Obr. 53.1.: Služby, poskytované na jednotlivých úrovních síťového modelu TCP/IP

Nad touto nespolehlivou nespojovanou přenosovou službou na úrovni síťové vrstvy pak TCP/IP model buduje spolehlivou transportní službu, a nad ní pak různé druhy aplikačních služeb - viz obrázek 53.1.

IP, neboli Internet Protocol

Věnujme se nyní způsobu, jakým je v TCP/IP modelu implementována výše zmíněná nespolehlivá a nespojovaná přenosová služba na úrovni síťové vrstvy. Definuje ji protokol, který se plným jménem nazývá **Internet Protocol**, ale nejčastěji je označován zkratkou **IP.**

IP protokol definuje základní jednotku dat, která je soustavou TCP/IP sítí přenášena na úrovni síťové vrstvy, tzv. **Internet datagram**, zkráceně **IP datagram**, a jeho přesný vnitřní formát. Definuje také způsob, jakým mají být jednotlivé IP datagramy směrovány, a toto směrování pak příslušný IP software také zajišťuje. Kromě toho IP protokol také podrobněji definuje i další aspekty poskytované nespolehlivé a nespojované přenosové služby - například podmínky, za jakých mohou být přenášené pakety zahazovány, kdy mají být generována chybová hlášení a jaká tato hlášení mají být.

.PI OBR53_2.TIF, 20, 40, 10

Obr. 53.2.: Představa IP datagramů a datových rámců

IP datagram a jeho formát

Podobně jako každý jiný druh paketu či rámce, má i IP datagram dvě základní části: řídící část, tvořenou hlavičkou datagramu, a datovou část. Jak naznačuje obrázek 53.2., při vlastním přenosu se tento datagram vkládá (jako data) do datové části rámce, se kterým pracuje bezprostředně nižší vrstva - vrstva síťového rozhraní. V hlavičce jsou pak různé řídící informace, potřebné pro doručení datagramu resp. rámce - v případě hlavičky rámce jde mj. o fyzickou adresu skutečného odesilatele a bezprostředního příjemce, zatímco v hlavičce IP datagramu jde o IP adresy koncového příjemce a původního odesilatele. Jelikož v datovém rámci mohou být v principu přenášeny i jiné druhy paketů, musí zde být vyjádřeno také to, o jaký konkrétní druh paketu se v daném případě jedná (například v sítích typu Ethernet se druh paketu udává pomocí dvoubytové položky v hlavičce rámce, a IP datagramy zde mají vyhrazen kód 0800H). Podobně je tomu i na úrovni paketu, resp. IP datagramu - také v jeho hlavičce musí být vyjádřeno, jaký je význam obsahu datové části. V případě IP datagramu jde o jednobytovou položku, udávající číslo protokolu na úrovni transportní vrstvy, kterému obsah IP datagramu patří (položka PROTOCOL, viz obr. 53.3.).

.PI OBR53_3.TIF, 20, 40, 10

Obr. 53.3.: Formát hlavičky IP datagramu

Maximální délka IP datagramu

Konkrétní formát hlavičky IP datagramu ukazuje obrázek 53.3. Za zmínku stojí například velikost položky TOTAL LENGTH, která udává celkovou délku IP datagramu (tj. jeho hlavičky i datové části), měřenou v oktetech (tj. v osmibitových bytech). Jelikož tato položka má rozsah 16 bitů, omezuje tím maximální možnou velikost IP datagramu na 65 535 oktetů.

Doba života

Další zajímavou položkou je jednobytová položka TIME TO LIVE. Ta udává, jak dlouho (měřeno v sekundách) se daný IP datagram může nacházet v soustavě vzájemně propojených sítí. Tato položka je jakousi pojistkou proti nekonzistentnostem ve směrovacích tabulkách, díky kterým by mohlo dojít k situaci, že by IP datagram byl směrován v kruhu, a obíhal v něm nekonečně dlouho. Tomu zabraňuje právě tato položka, jejíž obsah je každá mezilehlá brána povinna snižovat podle toho, jak dlouho se v ní příslušný datagram "zdrží". Jakmile dojde k vynulování položky TIME TO LIVE (doslova: doba života), je podle pravidel IP protokolu brána oprávněna daný IP datagram zahodit.

Fragmentace datagramů

Skutečnost, že IP protokol musí být schopen přenášet své datagramy prostřednictvím různých druhů přenosových cest, má některé významné důsledky. Například velikost datových rámců, přenášených na úrovni vrstvy síťového rozhraní TCP/IP modelu, je závislá na konkrétní přenosové technologii, pomocí které je daná dílčí síť realizována. V případě lokálních sítí typu Ethernet je to 1500 oktetů, zatímco například sítě Token Bus připouští rámce až do velikosti 8191 oktetů, a veřejné datové sítě na bázi doporučení X.25 pracují s rámci až do velikosti 4096 oktetů. Některé moderní přenosové technologie však pracují s mnohem menšími rámci - například jen 128 oktetů či ještě méně. Protokol IP se ale dost dobře nemůže přizpůsobit nejmenšímu možnému formátu rámce, aby do něj mohl vždy vložit svůj IP datagram celý. Proto musí počítat s možností fragmentace (viz též 41. díl našeho seriálu), při které pro potřeby přenosu dochází k rozdělení původního datagramu na několik dílčích fragmentů - tak velkých, aby se již vešly celé do těch rámců, které je příslušná síť schopna skutečně přenést - viz obr. 53.4. Jak jsme si již také naznačili ve 41. dílu, jde přitom o tzv. netransparentní fragmentaci, při které jednotlivé fragmenty skládá do původního celku až jejich koncový příjemce. Ten pak k tomu využívá položky IDENTIFICATION, FLAGS a FRAGMENT OFFSET jednotlivých fragmentů.

Nepovinné části IP datagramů

Položka IP Options, která v hlavičce IP datagramu následuje IP adresu konečného adresáta, je položkou nepovinnou. Je určena především pro potřeby testování a ladění. Pomocí této položky, která má proměnnou délku, je například možné zaznamenávat cestu, kterou je příslušný datagram postupně přenášen - což je neocenitelné zvláště pro potřeby správy sítí. Kromě IP adres bran, kterými datagram postupně prochází, je možné v této položce zaznamenávat i čas, kdy datagram jednotlivými bránami prochází. Dalším mechanismem, velmi užitečným pro správu vzájemně propojených sítí, je pak možnost explicitně předepsat v této položce trasu, kterou má být IP datagram doručen.

.PI OBR53_4.TIF, 20, 40, 10

Obr. 53.4.: Představa fragmentace

54/ Transportní protokoly TCP/IP - I.

Při srovnávání celkové filosofie síťového modelu TCP/IP a referenčního modelu ISO/OSI jsme si již dříve uvedli, že jednou z jejich odlišností je požadavek na spolehlivost přenosových služeb na jednotlivých úrovních. Referenční model ISO/OSI (alespoň ve své původní koncepci) je orientován především na spojované služby spolehlivého charakteru, a předpokládá tudíž zařazení potřebných mechanismů pro zajištění spolehlivosti již do vrstvy síťové. Naproti tomu TCP/IP model považuje zajištění spolehlivosti za úkol koncových účastníků, a zařazuje proto příslušné mechanismy až do vrstvy transportní.

Pro správné pochopení podstaty celého problému je třeba si uvědomit, že realizace spolehlivých služeb je spojena s větší režií, než jakou jsou zatíženy nespolehlivé přenosové služby. Nejde přitom jen o větší výpočetní náročnost, ale také o větší nároky na přenosovou kapacitu. Spolehlivost přenosů po ne zcela spolehlivých přenosových cestách se totiž musí zajišťovat vhodnou formou potvrzování (viz 30. díl seriálu), což je ale vždy spojeno s přenosem určitého objemu služebních dat, ke kterému samozřejmě dochází na úkor "užitečných dat".

Jestliže tedy TCP/IP model požaduje na síťové vrstvě pouze nespolehlivou přenosovou službu (realizovanou protokolem IP, viz minulý díl), umožňuje tím současně, aby síťová vrstva pracovala s menší režií, a tudíž rychleji.

Spolehlivost vs. rychlost

Považuje-li síťový model TCP/IP otázku zajištění spolehlivosti za úkol koncových účastníků komunikace, znamená to, že příslušné mechanismy pro její zajištění musí být implementovány ve vyšší vrstvě, než je vrstva síťová. Otázkou ovšem je, zda to skutečně musí být již vrstva bezprostředně vyšší, tedy vrstva transportní - jak jsme dosud předpokládali.

Zařadit potřebné mechanismy pro zajištění spolehlivosti do transportní vrstvy má četné výhody. Především pak tu, že tyto prostředky mohou být v transportní vrstvě realizovány právě jednou, a sdíleny všemi entitami aplikační vrstvy. Pokud bychom naopak tyto prostředky umístili až do vrstvy aplikační, vlastně by to znamenalo, že by si je musela každá entita aplikační vrstvy zajišťovat vždy znovu a sama.

Mohlo by se tedy zdát, že vše hovoří ve prospěch první možnosti. Ale i ta druhá může mít své opodstatnění - existují totiž i takové druhy aplikací, které buď vůbec nepotřebují spolehlivé přenosové služby, nebo naopak nepovažují "spolehlivost" na úrovni transportní vrstvy za dostatečnou, a tak si ji musí zajišťovat samy a znovu. Jinou motivací mohou být otázky efektivnosti - některé aplikace si dokáží zajistit samy takovou míru spolehlivosti, jakou skutečně potřebují, a vykazují přitom menší režii, než jakou na zajištění "úplné" spolehlivosti vyžaduje vrstva transportní. Příkladem takovéhoto druhu aplikací může být distribuovaný systém souborů NFS (Network File System) firmy Sun, velmi oblíbený a značně rozšířený v prostředí operačního systému Unix. Umožňuje, aby jeden uzlový počítač zcela transparentně sdílel soubory jiného uzlového počítače, tedy aby k nim přistupoval naprosto stejně, jako ke svým vlastním souborům. Nemá-li ale docházet k výraznějším časovým rozdílům při přístupu k lokálním a vzdáleným souborům, musí systém NFS pracovat

maximálně rychle. Nemůže si proto dovolit používat spolehlivé, zato ale "pomalé" přenosové služby na úrovni transportní vrstvy.

.PI OBR54_1.TIF, 20, 40, 10

Obr. 54.1.: Využití transportních protokolů některými protokoly aplikační vrstvy TCP/IP

Každý si může vybrat sám

Síťový model TCP/IP vychází vstříc oběma skupinám aplikací - jak těm, které na transportní vrstvě požadují spolehlivý přenos, tak i těm, které jej chtějí mít co možná nejrychlejší. Součástí rodiny protokolů TCP/IP jsou totiž dva alternativní protokoly pro transportní vrstvu: protokol **TCP** (**Transmission Control Protocol**), který zajišťuje spolehlivou přenosovou službu, a protokol **UDP** (**User Datagram Protocol**), který naopak nabízí službu nespolehlivou, s výrazně nižší režií, a tudíž rychlejší. Jednotlivé protokoly aplikační vrstvy si pak mezi oběma protokoly transportní vrstvy mohou vybrat ten, který jim lépe vyhovuje, případně využívat oba - několik konkrétních příkladů naznačuje obrázek 54.1.

Spojovaný vs. nespojovaný charakter transportních služeb

Spolehlivost a nespolehlivost není jediným rozdílem mezi oběma protokoly transportní vrstvy. Protokol TCP má spojovaný (connection-oriented) charakter, který pracuje s virtuálními okruhy (virtual circuits), a před vlastním přenosem předpokládá navázání spojení mezi odesilatelem a příjemcem. Naproti tomu protokol UDP má nespojovaný (connectionless) charakter, a každý jednotlivý blok dat, označovaný v tomto případ jako uživatelský datagram (user datagram) přenáší samostatně a nezávisle na ostatních datagramech.

Protokol UDP tak vlastně poskytuje na úrovni transportní vrstvy služby stejného charakteru, jaké na úrovni vrstvy síťové poskytuje protokol IP (viz minule), tedy nespolehlivé a nespojované služby. Naproti tomu protokol TCP implementuje pomocí nespolehlivých a nespojovaných služeb na úrovni síťové vrstvy spolehlivé a spojované transportní služby. Jak jsme si již naznačili výše, činí tak s pomocí mechanismu potvrzování (acknowledgement), a tudíž i s nezanedbatelnou režií.

Proud vs. blokově orientovaný přenos

Další odlišností mezi oběma protokoly je pak také forma, jakou své přenosové služby nabízí. Protokol UDP koncipuje svou přenosovou službu jako přenos celých bloků dat. Očekává tedy, že entita aplikační vrstvy, která chce odeslat nějaká data, je sama sestaví do bloku, a ten pak protokolu UDP předá jako jeden celek. Protokol UDP tento blok vezme, umístí jej do svého uživatelského datagramu (viz výše), a přenese opět jako jeden celek. Naproti tomu protokol TCP koncipuje svou přenosovou službu jako tzv. **proud** (**stream**) jednotlivých bytů (přesněji: osmic bytů, tzv. oktetů). Odesilatel na jednom uzlovém počítači postupně předává protokolu TCP jednotlivé byty (oktety), a obdobně postupuje i příjemce, který si je na druhé straně zase postupně odebírá. Veškeré členění přenášených dat na bloky a přenos celých těchto

bloků zajišťuje protokol TCP plně transparentně. Uživatelé přenosových služeb protokolu TCP tak získávají iluzi, že pracují se službou, která přenáší individuálně jednotlivé byty (oktety).

Stavový vs. bezestavový způsob komunikace

Se spojovaným, resp. nespojovaným charakterem protokolů TCP a UDP úzce souvisí ještě i další aspekt, který je velmi podstatný pro protokoly aplikační vrstvy a pro jejich volbu mezi oběma nabízenými transportními protokoly.

Každý uživatelský datagram (user datagram) - jak je označován blok dat, přenášený na úrovni transportní vrstvy protokolem UDP - je považován za samostatný celek, a vzhledem k nespojovanému charakteru protokolu UDP je také tak přenášen - samostatně, nezávisle na jiných uživatelských datagramech, a bez předchozího navázání spojení mezi příjemcem a odesilatelem. Stejně tak i příjemce chápe uživatelský datagram jako jediný celek, a ne jako součást většího celku. Tomu pak odpovídá i skutečnost, že příjemce nemění svůj stav v závislosti na průběhu komunikace - po přijetí datagramu se nachází ve stejném stavu, v jakém byl před jeho přijetím. Jde tedy o způsob přenosu, který je možné charakterizovat jako **bezestavový** (**stateless**).

Naproti tomu každá spojovaná služba má vždy nutně stavový charakter, neboť již pouhým navázáním spojení se příjemce dostává do jiného stavu, než v jakém byl předtím.

Bezestavový způsob komunikace je vzhledem ke své podstatě dobře odolný vůči různým nestandardním situacím, především pak výpadkům a ztrátám dat. Dojde-li například k výpadku přenosových cest či k výpadku příjemce (a je nutné jej znovu spustit, provést tzv. reboot), nejsou nutná žádná specifická opatření pro obnovu původního stavu - v přenosu dat se jednoduše pokračuje dál. Naproti tomu v případě stavového způsobu komunikace mohou být určité nápravné akce zapotřebí, má-li být zachována konzistence právě probíhajících činností, a znovu navázáno přerušené spojení.

S tím pak také souvisí i otázka spolehlivosti - zatímco bezestavové protokoly si mohou dovolit být nespolehlivé (jako UDP), pro jejich stavové protějšky je výhodnější pracovat s dostatečnou mírou spolehlivosti, a tedy nabízet spolehlivé přenosové služby (jako protokol TCP).

55/ Transportní protokoly TCP/IP - II.

V minulém dílu našeho seriálu jsme se začali zabývat transportní vrstvou síťového modelu TCP/IP. Seznámili jsme se se dvěma alternativními protokoly - TCP a UDP - které jsou na této úrovni k dispozici, a zabývali jsme se především jejich odlišnostmi. Dnes se podrobněji zastavíme u toho, co mají oba transportní protokoly společného.

Až doposud jsme v našich úvahách předpokládali, že prvotními odesilateli a konečnými příjemci nejrůznějších dat v počítačových sítích jsou jednotlivé uzlové počítače jako takové. Nyní si však musíme tuto představu upřesnit: skutečnými odesilateli a příjemci jsou entity aplikační vrstvy. Tyto mohou být označovány různě: jako procesy (processes), úlohy (tasks), aplikační programy (application programs) či ještě jinak. Podstatné ovšem je, že v prostředí víceúlohových operačních systémů

takovýchto entit může být (a také bývá) více. Oba protokoly transportní vrstvy pak potřebují podle něčeho rozhodnout, které entitě aplikační vrstvy mají přijatá data předat.

Adresy procesů?

Nejjednodušší by bylo adresovat jednotlivá data přímo konkrétním procesům. Tedy přidělit procesům jednoznačné identifikátory, a ty pak používat pro určení koncových příjemců v rámci příslušného hostitelského počítače. Problém je ovšem v tom, že procesy vznikají a zanikají dynamicky, a odesilatel obvykle nemá dostatek informací o tom, které konkrétní procesy právě běží na cílovém počítači. Kdyby tomu tak mělo být, při vzniku či zániku každého procesu by museli být průběžně informováni všichni potenciální odesilatelé, což není prakticky možné. Pro odesilatele navíc nemusí být vůbec podstatné, který konkrétní proces bude příjemcem jeho dat. Data, která proces na jednom počítači posílá na jiný počítač, totiž mohou představovat požadavek na určitou službu. Pak ale není relevantní to, který proces právě zajišťuje danou službu, jako spíše to, o jakou službu jde. Navíc je docela dobře možné, že jeden a tentýž proces zajišťuje více jak jednu službu současně.

Adresovat data konkrétním procesům tedy není příliš rozumné. Jaké jsou ale alternativy?

Porty a jejich vlastnosti

Další možností je vytvořit na rozhraní mezi transportní vrstvou a bezprostředně vyšší vrstvou "přechodové body", a ty opatřit jednoznačnými identifikátory, resp. adresami. Přenášená data pak jsou adresována těmto "přechodovým bodům", a skutečným příjemcem dat je pak ten proces, který je v daném okamžiku k tomuto "přechodovému bodu" logicky připojen.

Síťový model TCP/IP používá právě tento mechanismus, a příslušné "přechodové body" označuje jako **porty** (**ports**).

Z pohledu procesu, který je právě připojen k určitému portu, se tento jeví jako vyrovnávací paměť s režimem fronty - entita transportní vrstvy ukládá data do portu z jedné strany, a entita aplikační vrstvy (tj. proces) si je ve stejném pořadí zase odebírá - viz obr. 55.1. Analogicky i pro opačný směr přenosu. Samotný port s právě naznačenými vlastnostmi je sám o sobě datovou strukturou, a jako takový může dynamicky vznikat i zanikat, obdobně jako vlastní procesy.

.PI OBR55_1.TIF, 20, 40, 10

Obr. 55.1.: Představa portů

Adresy portů

Vztah mezi portem a procesem (entitou aplikační vrstvy) je vztahem dynamickým - procesy se dynamicky připojují a odpojují od jednotlivých portů. Pro odesilatele je tedy podstatné znát čísla (adresy) jednotlivých portů, a nemusí pro ně být relevantní, který konkrétní proces je k příslušnému portu právě připojen. Chce-li například určitý

proces na hostitelském počítači A získat soubor, který se nalézá na počítači B, musí svůj požadavek na tento soubor adresovat počítači B (který určí jeho IP adresou), a doplnit jej adresou portu, na kterém je zajišťována služba pro přenos souborů. Adresy portů jsou přitom celými čísly bez znaménka, v rozsahu 16 bitů.

Otázkou ovšem je, jak může aplikační proces na jednom hostitelském počítači zjistit potřebná čísla portů na jiném počítači, ať již chce využít některou z jím poskytovaných služeb, nebo jinak komunikovat s některým z aplikačních procesů na tomto jiném počítači.

K řešení této otázky jsou v zásadě dva možné přístupy. První z nich spočívá v tom, že čísla portů se na všech hostitelských počítačích přidělí jednou provždy a stejně, a toto přidělení se zveřejní. Každý aplikační proces pak má k dispozici informaci o číslech příslušných portů na všech uzlových počítačích - těmto portům se pak také říká **dobře známé porty** (**well-known ports**). Problém je ovšem v tom, že tato varianta statického charakteru se dá použít jen pro zveřejnění takových služeb, které jsou předem známy, a s jejichž existencí lze předem počítat.

Alternativou je umožnit procesům na jiných uzlových počítačích zjistit si číslo portu na daném počítači - dotazem stylu: "na jakém portu je poskytována služba přenosu souborů?" Tato varianta stále vyžaduje alespoň jeden "dobře známý port" (přes který by bylo možné vznést příslušný dotaz), a je také spojena s určitou režií. Na druhé straně je ale mnohem pružnější, neboť umožňuje vytvářet porty a přidělovat jim čísla (adresy) podle okamžité potřeby dynamicky vznikajících a zanikajících aplikačních procesů, a není vázána tím, jak jsou obdobná přidělení realizována na jiných uzlových počítačích.

Tvůrci TCP/IP protokolů se rozhodli pro kombinaci obou variant - pro nejčastěji používané služby použili první možnost, tedy pevné přidělení čísel "dobře známým portům", a zbylé adresy ponechali volné pro dynamické přidělování podle okamžitých potřeb.

Multiplex a demultiplex datagramů

Vraťme se nyní zpět k oběma protokolům transportní vrstvy soustavy TCP/IP, tedy k protokolům UDP a TCP. Z existence více portů mezi transportní a aplikační vrstvou vyplývá, že protokol transportní vrstvy na straně odesilatele musí být schopen přijímat data od různých aplikačních entit, a předávat je síťové vrstvě (realizované pomocí protokolu IP) k přenosu - tedy zajišťovat jejich multiplexování, a na straně příjemce pak provádět jejich demultiplexování, tedy rozdělování dat, převzatých od síťové vrstvy, mezi různé entity aplikační vrstvy, prostřednictvím příslušných portů (viz obrázek 55.2).

20

.PI OBR55_2.TIF, 20, 40, 10

Obr. 55.2.: Multiplex a demultiplex datagramů

Tato představa velmi dobře odpovídá koncepci nespojované přenosové služby, kterou poskytuje protokol UDP. Tento protokol se na každý port dívá skutečně jako na jeden

logický objekt - frontu, do které jsou postupně ukládány všechny datagramy, adresované tomuto portu na daném počítači.

Protokol TCP však vychází z poněkud odlišné abstrakce. Vzhledem ke svému spojovanému charakteru pracuje se spojeními, která jsou definována dvěma koncovými body - každý koncový bod je přitom určen dvojicí <IP adresa počítače, číslo portu>. Více různých spojení přitom může sdílet tentýž koncový bod, tj. "končit" ve stejném portu. Základním typem objektu, se kterým protokol TCP pracuje, je tedy spojení, a nikoli port jako takový.

56/ Protokol UDP

V předchozích dílech našeho seriálu jsme se začali zabývat transportní vrstvou síťového modelu TCP/IP. Přitom jsme si naznačili, že pro tuto vrstvu připadají v úvahu dva alternativní protokoly - TCP a UDP. Dnes se podrobněji zastavíme u jednoho z nich: protokolu UDP.

Nejprve si ale znovu připomeňme některé obecnější souvislosti, o kterých jsme zmínili již v 53. dílu našeho seriálu. Zde jsme si řekli, že na rozdíl od referenčního modelu ISO/OSI považuje síťový model TCP/IP otázku zabezpečení spolehlivosti přenosů za věc koncových účastníků komunikace, a z tohoto důvodu zařazuje příslušné mechanismy pro zajištění spolehlivosti až do transportní vrstvy, zatímco na úrovni síťové vrstvy předpokládá pouze nespolehlivou službu nespojovaného charakteru (realizovanou protokolem IP). Řekli jsme si však také, že síťový model TCP/IP pamatuje i na to, že pro některé druhy aplikací nemusí být spolehlivé transportní služby výhodné. Proto jim nabízí jako alternativní možnost využívat i na úrovni transportní vrstvy služby stejné povahy a kvality, jakou mají přenosové služby na úrovni síťové vrstvy, zajišťované protokolem IP - tedy nespolehlivé přenosové služby nespojovaného charakteru.

UDP jako jednoduchá "obálka" protokolu IP

Mechanismem, který entitám (procesům, úlohám, programům) aplikační vrstvy zpřístupňuje nespolehlivé a nespojované, zato ale rychlé a efektivní přenosové služby síťové vrstvy (protokolu IP), je právě protokol **UDP** (**User Datagram Protocol**). Můžeme si jej představit jako jednoduchou "obálku" nad protokolem IP, která nijak nemění povahu ani kvalitu jeho přenosových služeb, ale pouze je zprostředkovává své bezprostředně vyšší vrstvě. V podstatě jediné, co UDP zajišťuje navíc, je multiplexování a demultiplexování datagramů (viz minulý díl seriálu), tedy rozlišování různých příjemců, resp. odesilatelů v rámci téhož hostitelského počítače - podle čísla portu.

Odpovědnost přebírá aplikace

Každý aplikační program, který se rozhodne používat transportní služby protokolu UDP, tak na sebe přebírá odpovědnost za zajištění takové úrovně spolehlivosti přenosů, jakou sám potřebuje. Sám se také musí vyrovnávat i s dalšími důsledky, které vyplývají z nespolehlivého a nespojovaného charakteru přenosových služeb

protokolu UDP - jako je např. zajišťování správného pořadí doručovaných datagramů, eliminace duplicitních datagramů apod.

Jak jsme si již uvedli v předchozích dílech, je volba nespolehlivých transportních služeb protokolu UDP determinována především charakterem aplikace a jeho požadavky na efektivitu přenosových služeb. Svou roli při rozhodování mezi nespolehlivými ale rychlejšími službami protokolu UDP a spolehlivými, zato ale pomalejšími transportními službami protokolu TCP však sehrává i prostředí, ve kterém jsou tyto transportní služby zajišťovány. Například v lokálních sítích, které jsou velmi rychlé a především relativně spolehlivé, může být protokol UDP pro mnohé aplikace velmi výhodný. Při přechodu do prostředí rozlehlých sítí, které jsou ve své podstatě mnohem méně spolehlivé, však může natolik narůst režie konkrétního aplikačního programu na zajištění potřebné spolehlivosti, že se pro ni stane protokol UDP méně výhodný, než "spolehlivý" protokol TCP, nebo se použití protokolu UDP stane dokonce zcela neúnosné.

.PI OBR56_1.TIF, 20, 40, 10

Obr. 56.1.: UDP datagramy vs. IP datagramy

Formát UDP datagramu

Jak jsme si již také uvedli v 53. dílu, protokol UDP dostává od své bezprostředně vyšší vrstvy bloky dat, které se snaží vkládat celé do jednotlivých datagramů síťové vrstvy (IP datagramů), viz obr. 56.1. Proto se také těmto blokům na úrovni transportní vrstvy říká **uživatelské datagramy** (**user datagrams**), nebo též UDP datagramy (UDP datagrams). Jejich formát je uveden na obrázku 56.2.

20

Hlavičku (anglicky: header) UDP datagramu tvoří 4 položky v rozsahu 16 bitů, které po řadě vyjadřují číslo portu odesilatele a příjemce, délku UDP datagramu, a kontrolní součet - viz obr. 56.2.

.PI OBR56_2.TIF, 20, 40, 10

Obr. 56.2.: Formát UDP datagramu

Číslo portu příjemce (položka UDP DESTINATION PORT) je základní informací, podle které se protokol UDP na straně příjemce rozhoduje, komu má přijatý datagram doručit - přesněji: přes který port má přijatý datagram předat entitě aplikační vrstvy. Číslo portu odesilatele (v položce UDP SOURCE PORT) je nepovinné; vyplňuje se v případě, kdy je požadována odpověď (v opačném případě se tato položka zaplňuje nulami).

Položka LENGTH vyjadřuje délku UDP datagramu, měřenou v oktetech (tj. osmicích bitů) - včetně vlastní hlavičky. Minimální délka UDP datagramu je proto 8, což je právě velikost hlavičky, s prázdnou datovou částí.

Kontrolní součet

Kontrolní součet (položka CHECKSUM) je počítán v rozsahu 16 bitů v aritmetice, která pro vyjádřená čísel se znaménkem využívá tzv. jedničkový doplněk (one's

complement). Jednou ze zajímavých vlastností této nepříliš používané aritmetiky je skutečnost, že má dva možné způsoby znázornění nuly: tzv. kladnou nulu, tvořenou dvojkově samými nulami, a tzv. zápornou nulu, která má naopak všechny bity nastaveny na jedničku. V případě, že kontrolní součet vychází nulový, je použita právě tato druhá možnost, tedy tzv. záporná nula. Použití kontrolního součtu však není povinné. Jestliže se kontrolní součet nepoužije, je položka CHECKSUM vynulována - ovšem tzv. kladnou nulou, což je díky vlastnostem jedničkového doplňku možné odlišit od nulové hodnoty kontrolního součtu.

Zvláštností protokolu UDP je skutečnost, že zmíněný kontrolní součet - je-li použit - není počítán z UDP datagramu (který je na obrázku 56.2.), ale z větší datové struktury (viz obr. 56.3.), která vznikne pomyslným připojením další hlavičky - tzv. **pseudohlavičky** (**pseudo header**) k vlastnímu datagramu. Pomyslným připojením v tom smyslu, že tato pseudohlavička není ve skutečnosti přenášena od odesilatele k příjemci, ale je brána v úvahu pouze pro výpočet kontrolního součtu.

.PI OBR56_3.TIF, 20, 40, 10

Obr. 56.3.: UDP datagram a pseudohlavička

Jak vyplývá z obrázku 56.3., je v pseudohlavičce obsažena mj. IP adresa příjemce a odesilatele (která naopak ve vlastním UDP datagramu, resp. jeho hlavičce obsažena není). Protokol UDP na straně příjemce kontroluje bezchybovost všech přijatých datagramů novým výpočtem kontrolního součtu a jeho porovnáním s obsahem položky CHECKSUM. Uživatelský datagram sice přijme bez jeho pseudohlavičky, ale pro potřeby kontrolního součtu si ji protokol UDP znovu vytvoří a zahrne ji do nového výpočtu kontrolního součtu. Vzhledem k tomu, že pseudohlavička UDP datagramu obsahuje mj. i IP adresu příjemce, jde o mechanismus, který kromě poškozených datagramů umožňuje rozpoznat také datagramy, doručené na chybnou IP adresu (což z pouhého čísla portu, které je ve vlastním datagramu, nelze poznat), a následně je eliminovat.

Zastavme se však ještě u právě naznačeného výpočtu kontrolního součtu. Podle něj musí protokol UDP u odesilatele již při sestavování UDP datagramu znát nejen IP adresu příjemce, ale i IP adresu odesilatele, neboť obě tyto adresy musí zahrnout do kontrolního součtu. IP adresu příjemce se protokol UDP dozví od aplikace, neboť právě ona určuje, komu jsou příslušná data určena. Pokud však jde o IP adresu odesilatele, nemusí být v silách protokolu UDP ji stanovit. Jde-li totiž o hostitelský počítač, který má více než jedno síťové rozhraní (tj. jde-li o tzv. multi-homed host, který je současně zapojen do více dílčích sítí, viz 46. díl seriálu), má odesilatel více různých IP adres (po jedné na každé síťové rozhraní, resp. pro každou dílčí síť). Rozhodnout, kudy bude příslušný datagram skutečně odeslán (tj. kterým síťovým rozhraním, resp. s jakou IP adresou odesilatele), je pak až v silách síťové vrstvy, resp. protokolu IP, který zajišťuje směrování. Protokol UDP si proto musí při sestavování uživatelského datagramu vyžádat pomoc protokolu IP, což ale poněkud narušuje standardní způsob interakce jednotlivých vrstev ve vrstevnatém modelu.

57/ Protokol TCP - I.

Řekne-li se dnes TCP/IP, mnoho lidí si pod tím představí dvojici protokolů: TCP (Transmission Control Protocol) a IP (Internet Protocol). Z našeho dosavadního povídání o počítačových sítích však již víme, že ve skutečnosti jde o

celou soustavu protokolů (anglicky: protocol suite), spojenou s vlastní soustavou názorů na to, jak by počítačové sítě měly vypadat a jak by měly fungovat - tedy to, co jsme si již dříve označili jako síťovou architekturu či síťový model. Protokol TCP je tedy jen jedním z více protokolů soustavy TCP/IP, i když protokolem velmi významným. Dnes se u něj zastavíme podrobněji.

Nejprve si ale znovu připomeňme, že protokol TCP je protokolem transportní vrstvy, a že je jedním ze dvou alternativních protokolů, které síťový model TCP/IP na úrovni této vrstvy nabízí. Připomeňme si také důvod této nabídky dvou alternativních protokolů: jednotlivé aplikace mají možnost si vybrat, zda je pro ně výhodnější používat na úrovni transportní vrstvy nespolehlivé (ale rychlejší) přenosové služby, zajišťované protokolem UDP (User Datagram Protocol), nebo naopak pomalejší, zato ale spolehlivé služby protokolu TCP. V minulém dílu našeho seriálu jsme se podrobněji zabývali protokolem UDP a naznačili jsme si, že je vlastně jen jednoduchou obálkou nad přenosovými službami síťové vrstvy, která nijak nemění jejich kvalitu (tj. nespolehlivost) ani povahu (nespojovaný charakter), a pouze je zpřístupňuje entitám aplikační vrstvy. Úkol protokolu TCP je z tohoto pohledu mnohem náročnější - k dispozici má stejné nespolehlivé služby na úrovni síťové vrstvy, ale s jejich pomocí musí sám zajišťovat služby spolehlivé.

Další významný rozdíl mezi protokoly UDP a TCP spočívá v tom, že zatímco UDP nabízí nespojované (connectionless) služby, protokol TCP nabízí své přenosové služby jako spojované (connection-oriented). Před každou výměnou dat mezi dvěma uzly tedy musí být nejprve navázáno spojení, a po skončení přenosu musí být toto spojení zase zrušeno. Výhodami i nevýhodami spojovaných a nespojovaných služeb jsme se zabývali již ve 27. dílu našeho seriálu.

Stream, neboli proud

Dalším významným rozdílem mezi přenosovými službami protokolů TCP a UDP je také jejich pohled na přenášená data. Jak jsme si již naznačili minule, protokol UDP očekává od své bezprostředně vyšší vrstvy vždy celý blok dat, který se snaží přenést opět jako celek (v rámci jediného tzv. uživatelského datagramu, viz minule), a na straně příjemce jej předává své bezprostředně vyšší vrstvě opět jako celek.

Naproti tomu protokol TCP se snaží nabízet své bezprostředně vyšší vrstvě přenos jednotlivých bytů. Očekává tedy, že entity aplikační vrstvy mu na straně odesilatele budou postupně předávat jednotlivé byty (přesněji: osmibitové oktety), a na straně příjemce si je zase budou po jednom vyzvedávat. Tím vzniká iluze **proudu** (anglicky: **stream**) jednotlivých bytů, který navíc není nijak strukturován - tj. všechny přenášené byty jsou považovány za rovnocenné. Ve skutečnosti samozřejmě nejsou jednotlivé bity přenášeny každý zvlášť. Protokol TCP na straně odesilatele postupně akumuluje jednotlivé byty do vhodné vyrovnávací paměti (bufferu), a po jejím naplnění odešle celý její obsah najednou - v tzv. **segmentu**, jak se nazývá blok, přenášený protokolem TCP. Analogicky na straně příjemce, kde se datový obsah segmentu ukládá do vyrovnávací paměti, a jednotlivé byty jsou entitám aplikační vrstvy poskytovány z této vyrovnávací paměti. Celý mechanismus sdružování jednotlivých bytů do bloků je plně v režii protokolu TCP, který se přenosem větších celků snaží optimalizovat využití přenosových cest. Pro vyšší vrstvu je tento mechanismus neviditelný - vyší vrstva pracuje s představou proudu jednotlivých bytů.

Existují však takové aplikace, pro které právě naznačený mechanismus není příliš vhodný. Příkladem mohou být tzv. vzdálené terminálové relace, uskutečňované prostřednictvím sítě - kdy jeden uzlový počítač vystupuje v roli terminálu jiného počítače a jednotlivé znaky, zadané z jeho klávesnice, posílá ke zpracování tomuto druhému počítači (který mu zase posílá zpět vše, co má být zobrazeno na obrazovce terminálu). Kdyby v takovéto situaci protokol TCP čekal, až vstupní znaky naplní vyrovnávací paměť na straně odesilatele a teprve pak je skutečně odeslal, uživatel by hodně dlouho mačkal klávesy bez jakékoli odezvy. Protokol TCP však i s tímto počítá, a nabízí odesilateli mechanismus (označovaný jako *push*), kterým si aplikace může vynutit skutečné odeslání dat i v případě, že příslušná vyrovnávací paměť není ještě naplněna.

Jak se dosahuje spolehlivost

Již ve 30. dílu seriálu jsme si naznačili, že prakticky jedinou možností, jak pomocí nespolehlivé přenosové služby zajistit spolehlivé přenosy, je umožnit příjemci rozpoznat chybně přenesený blok, a vyžádat si jeho opětovné vyslání.

Ukázali jsme si také, že za tímto účelem se používá více různých způsobů tzv. **potvrzování** (**acknowledgement**), viz opět 30. díl. Protokol TCP používá tzv. **kladné potvrzování** (**positive acknowledgement**), což znamená, že potvrzuje úspěšně přijatá data, a na chybně přijatá data nereaguje nijak (ty pak odesilatel znovu vyšle na základě vypršení časového limitu, ve kterém očekával jejich kladné potvrzení). Ve své základní podobě tzv. **jednotlivého kladného potvrzování** (**stop&wait positive acknowledgement**), kdy odesilatel před odesláním každého dalšího bloku čeká na kladné potvrzení naposledy vyslaného bloku, je tento mechanismus značně neefektivní. Protokol TCP proto používá kladné potvrzování ve variantě tzv. **kontinuálního potvrzování** (**continuous acknowledgement**), známého též jako tzv. **metoda okénka** (**sliding window**). Podstata této varianty spočívá v tom, že odesilatel může odeslat další blok (resp. několik dalších bloků) ještě dříve, než dostane potvrzení o přijetí bloku předchozího. O tom, kolik bloků může takto vyslat "dopředu", rozhoduje velikost pomyslného okénka (viz obr. 30.4. ve 30. dílu).

Když jsme se ve 30. dílu zabývali různými metodami potvrzování, předpokládali jsme, že potvrzovanými jednotkami jsou celé bloky dat. V případě protokolu TCP se přenášené bloky dat označují jako **segmenty**, a mohou mít různou délku. V případě, že některý segment není přenesen bezchybně (resp. není kladně potvrzen do vypršení časového limitu), je jeho obsah vyslán znovu, a po něm také obsah všech následujících segmentů. Jde tedy o tzv. **opakování s návratem** (**Go-Back-N)**, viz opět 30. díl. Podstatné ale je, že při tomto opakovaném vysílání již jednou vyslaných dat nemusí být dodrženy původní velikosti segmentů - znovu vyslaný segment může být větší než segment, vyslaný původně. To ale činí potvrzování celých segmentů problematické. Proto jsou v protokolu TCP potvrzovanými jednotkami dat nikoli celé bloky (segmenty), ale jednotlivé byty (přesněji: oktety)!!

.PI OBR57_1.TIF, 20, 40, 10

Obr. 57.1.: Představa potvrzování v protokolu TCP

Konkrétní způsob implementace potvrzování v protokolu TCP využívá plně duplexního charakteru spojení na úrovni transportní vrstvy. Jak naznačuje obrázek 57.1., každý segment obsahuje kromě vlastních dat i údaj o pozici prvního bytu těchto

dat v rámci celého proudu bytů (tzv. **sequence number**). S využitím tohoto údaje pak příjemce původní proud bytů rekonstruuje. Jednotlivá kladná potvrzení se opět vztahují k celým bytům, a jsou vkládána do segmentů, přenášených v opačném směru, než potvrzovaná data. Jde tedy o tzv. **nesamostatné potvrzování** (anglicky: **piggybacking**), viz 30. díl. Kladné potvrzení má přitom formu čísla pozice prvního bytu (v rámci proudu), který příjemce očekává jako další (tj. prvního bytu za posledním úspěšně přijatým) - jde o tzv. **acknowledgement number**.

58/ Protokol TCP - II.

V minulém dílu jsme se začali zabývat protokolem TCP. Řekli jsme si, že své bezprostředně vyšší vrstvě poskytuje přenosové služby spolehlivého a spojovaného charakteru, i když k jejich zajištění může sám využívat jen nespolehlivé (a nespojované) přenosové služby na úrovni síťové vrstvy, poskytované protokolem IP. Naznačili jsme si také, že potřebné spolehlivosti dosahuje protokol TCP pomocí potvrzování a opakovaného vysílání chybně přenesených dat. Dnes se zastavíme podrobněji právě u způsobu opakování přenosů.

Připomeňme si však ještě jednou to, co jsme si o konkrétním mechanismu potvrzování řekli již minule: protokol TCP používá tzv. kladné potvrzování (positive acknowledgement), což znamená, že potvrzuje pouze správně přijatá data, zatímco na chybně přijatá data nereaguje nijak. Odesilatel proto pozná, že určitá data nebyla přenesena správně, až po vypršení určitého časového limitu. Současně s tím je ale dobré si uvědomit, že také kladná potvrzení jsou sama přenášena pomocí stejně nespolehlivých přenosových služeb síťové vrstvy, jako samotná data. Může se proto stát, že určitá data jsou přenesena bezchybně, ale "ztratí" se jejich kladné potvrzení. Odesilatel však nemá možnost oba tyto případy rozlišit. Vždy musí čekat až do vypršení časového limitu, zda nedostane kladné potvrzení, a pokud ne, musí data vyslat znovu.

Otázkou ovšem je, jak volit příslušný časový limit (**timeout**)? Bude-li příliš krátký, může docházet k situacím, kdy data budou přenesena bezchybně, ale jejich kladné potvrzení dojde odesilateli příliš pozdě - až po vypršení časového limitu, což odesilatele přinutí znovu vyslat úspěšně přenesená data. V opačném případě - bude-li časový limit příliš dlouhý - bude odesilatel zase zbytečně dlouho čekat, než si bude moci domyslet, že se data nepřenesla správně. V obou případech je výsledkem neefektivní využití přenosových kapacit.

Problém vhodné volby časového limitu je navíc komplikován skutečností, že protokol TCP může být používán v různých sítích, ve kterých se doby přenosu mohou lišit i v několika řádech. Například v rychlé lokální síti by příslušný časový limit měl být výrazně kratší než v rozlehlé síti, která používá relativně velmi pomalé pevné telefonní okruhy. Situace je navíc ještě komplikovanější u vzájemně propojených sítí (internets, s malým "i"), kde jednotlivé části přenosové trasy (route) mají obecně různé přenosové kapacity, a kde se navíc trasa mezi dvěma koncovými účastníky může v průběhu přenosu dynamicky měnit (např. v důsledku zahlcení či výpadku).

Adaptivní chování

Způsob, jakým se protokol TCP vyrovnává s problémem správné volby časového limitu pro opakované vysílání (retransmission), lze v prvním přiblížení charakterizovat jako adaptivní. TCP totiž průběžně monitoruje skutečné chování přenosové sítě, a podle něj se snaží předvídat její chování i pro nejbližší časový úsek. Tomu pak přizpůsobuje i své chování.

TCP neustále měří **dobu obrátky** (**RTT, round trip time**) jako dobu od odeslání dat do přijetí kladného potvrzení o jejich úspěšném doručení. Časový limit pak uzpůsobuje průměrné době obrátky, kterou vypočítává jako vážený průměr (weighted average) jednotlivých naměřených dob obrátky.

Nejednoznačnost potvrzování

V minulém dílu jsme si také říkali, že protokol TCP nepotvrzuje celé segmenty (jak se nazývají jím přenášené bloky dat), ale jednotlivé byty (přesněji: oktety) v rámci přenášeného proudu (stream). To má jeden závažný důsledek pro výpočet doby odezvy. Představme si situaci, kdy odesilatel nedostane do časového limitu kladné potvrzení o přijetí určité části dat, a tak je vyšle znovu ve stejně velkém segmentu. Dostane-li pak kladné potvrzení, není z něj schopen rozlišit, ke kterému segmentu se toto potvrzení vztahuje - zda jde o potvrzení poprvé vyslaných dat (které se někde zdrželo), nebo zda jde již o potvrzení podruhé vyslaných dat (zatímco poprvé vyslaná data se přenesla s chybou, nebo se jejich kladné potvrzení ztratilo). Potvrzování, používané protokolem TCP, je proto v obecném případě **nejednoznačné** (**ambiquous**).

Z pohledu zabezpečení spolehlivého přenosu tato nejednoznačnost nijak nevadí, ale problém vzniká při výpočtu doby obrátky: má tato být vztažena k okamžiku prvního vyslání, nebo naopak k okamžiku posledního opakovaného vyslání určitých dat? Dá se ukázat, že obě varianty jsou špatné - v případě té první může vážený průměr doby obrátky růst nade všechny meze, zatímco ve druhém případě sice konverguje k určité konečné hodnotě, ale tato je menší, než by správně měla být (praktické implementace této varianty ukazují, že je tato hodnota přibližně poloviční oproti správné, a protokol TCP pak vysílá všechna data právě dvakrát).

Karnův algoritmus

S jednoduchým a elegantním řešením poprvé přišel radioamatér Phil Karn, když se snažil implementovat TCP/IP protokoly i pro prostředí radioamatérských paketových sítí (což se mu ostatně povedlo - výsledkem je známý programový balík KA9Q, nazvaný podle volací značky svého autora).

Myšlenka, se kterou Karn přišel, je velmi jednoduchá - v případě nejednoznačného potvrzení (tj. v případě opakovaného vyslání již jednou vyslaných dat) vůbec neměřit dobu obrátky, a tudíž i neuzpůsobovat podle ní časový limit. Jinými slovy: pro stanovení časového limitu využívat pouze ty případy, kdy data byla vyslána jen jednou, a je tudíž jasné, jak dobu obrátky měřit.

I toto řešení však ve své čisté podobě není příliš použitelné. Představme si totiž situaci, kdy například vlivem změny trasy či výpadku části přenosových kapacit náhle vzroste zpoždění při přenosu. Potvrzení nestihne přijít včas, a tak jsou data vyslána znovu - v důsledku toho se pak ale neměří doba obrátky, a nijak se nemění ani časový

limit. Bude-li nárůst zpoždění trvalý, výše uvedený algoritmus na něj vůbec nezareaguje!

Konkrétní algoritmus, který Phil Karn navrhl, však řeší i tuto situaci (technikou, označovanou jako **timer backoff**): v okamžiku, kdy začne docházet k opakovanému vysílání dat, přestává tento algoritmus měřit dobu obrátky, a nemění tudíž ani vážený průměr těchto dob. Začne ale zvětšovat časový limit (např. na dvojnásobek po každém opakovaném vyslání dat). Teprve v okamžiku, kdy díky většímu časovému limitu dostane kladné potvrzení včas (tj. jako jednoznačné potvrzení), změří skutečnou dobu obrátky, znovu začne vypočítávat vážený průměr dob obrátky, a tomuto průměru pak uzpůsobí časový limit.

59/ Protokol TCP - III.

V minulém dílu jsme se zabývali tím, jak se protokol TCP vyrovnává s různou propustností přenosových cest, a jak dokáže dynamicky přizpůsobovat svůj mechanismus potvrzování okamžitým změnám tzv. doby obrátky. K tomu, aby protokol TCP dokázal co nejefektivněji využít existující přenosové kapacity, to ale ještě nestačí.

Dalším důležitým předpokladem pro zajištění spolehlivého, ale současně s tím i dostatečně efektivního přenosu, je vhodné **řízení toku** (**flow control**). Protokol TCP musí zajistit, aby odesilatel neposílal data rychleji, než je příjemce schopen je přijímat (resp. ukládat do svých vyrovnávacích pamětí). Pokud by totiž odesilatel příjemce "zahltil", tento by musel přijímaná data zahazovat, a ta by musela být posléze vysílána znovu. Odesilatel se tedy nutně musí řídit kapacitními možnostmi příjemce. Konkrétní způsob, jakým se v protokolu TCP dosahuje potřebného řízení toku, bezprostředně souvisí s mechanismem potvrzování.

Okénko proměnlivé velikosti

Když jsme si v 57. dílu popisovali mechanismus potvrzování v protokolu TCP,řekli jsme si, že jde o tzv. kladné potvrzování (tj. takové, kdy příjemce kladně potvrzuje správně přijatá data, a na chybně přenesená data nereaguje). Protokol TCP ovšem používá tento způsob potvrzování v tzv. kontinuální variantě, kdy umožňuje, aby odesilatel vysílal data "dopředu", tedy ještě dříve, než dostane potvrzení o úspěšném přijetí dříve vyslaných dat. Existuje zde samozřejmě určité omezení na objem dat, které se takto mohou vyslat "dopředu", a je dáno velikostí posuvného datového okénka (sliding window) - viz obr. 59.1. Okamžitá velikost a poloha tohoto okénka určuje odesilateli, kolik dat ještě může vyslat, aniž by musel čekat na potvrzení od protější strany.

Příjemce má samozřejmě možnost řídit tok dat tím, že dočasně pozdrží kladné potvrzení úspěšně přijatých dat. Tím totiž zabrání odesilateli, aby si posunul své datové okénko, a nedovolí mu tudíž vyslat další data za okamžitým koncem okénka. Tato metoda má ale nepříjemný vedlejší efekt v tom, že když příjemce pozdrží kladné potvrzení a odesilatel jej nedostane do příslušného časového limitu (viz minule), bude to interpretovat tak, že tato data nebyla úspěšně přenesena. V důsledku toho pak odesilatel znovu odešle již jednou odeslaná a úspěšně přijatá data. Tato sice již nemohou příjemce "zahltit" (ten je může jednoduše ignorovat), ale jejich přenos zbytečně spotřebovává část existující přenosové kapacity.

Protokol TCP proto používá jiný způsob řízení toku, kterým je změna velikosti posuvného okénka. Příjemce reaguje na každý úspěšný přenos dat tím, že pošle příslušné kladné potvrzení, a odesilatel si na základě tohoto potvrzení příslušným způsobem posune své datové okénko. Současně s tímto potvrzením ale příjemce zašle odesilateli ještě i další údaj, podle kterého si odesilatel nově nastaví šířku svého datového okénka (viz obr. 59.1.).

.PI OBR59_1.TIF, 20, 40, 10

obr. 59.1.: představa posuvného datového okénka

Příjemce tento údaj samozřejmě volí podle svých možností tak, aby vždy byl schopen přijmout to, co mu odesilatel pošle. Příjemce dokonce může odesilateli zmenšit jeho datové okénko až na nulovou velikost, a tím vlastně zcela zastavit vysílání jakýchkoli dat. Příjemce by ale při stanovování nové velikosti okénka měl vždy postupovat korektně - jelikož nemůže přesně určit okamžik, kdy odesilatel dostane údaj o nové velikosti okénka, neměl by se jeho pravý okraj nikdy posouvat zpět (na obr. 59.1 směrem doleva)!

Nejde jen o koncové účastníky

Potřeba řídit tok dat se však netýká jen obou koncových účastníků přenosu. Je totiž nutné si uvědomit, že na cestě od odesilatele k příjemci mohou data procházet i přes několik přepojovacích uzlů (gateways), které také mají jen konečnou propustnost. Odesilatel tedy musí dávat pozor i na to, aby nezpůsobil zahlcení (tzv. **zácpu**, angl. **congestion**) těchto mezičlánků.

Protokol TCP však neobsahuje žádný explicitní mechanismus, který by měl za úkol ochranu těchto mezičlánků před zahlcením (**congestion control**). Snaží se ale chovat tak, aby k zahlcování mezilehlých přepojovacích uzlů nedocházelo.

Pro správné pochopení podstaty celého problému je vhodné si uvědomit, jak se takové zahlcení mezilehlého uzlu projeví koncovým účastníkům komunikace. Jakmile přepojovací uzel (gateway) nestačí přepojovat přenášená data v reálném čase, tato se hromadí v jeho vstupních frontách, kde čekají na své další zpracování. Tento stav se koncovým účastníkům projevuje růstem doby obrátky. Jakmile ale zatížení přepojovacího uzlu stoupne natolik, že jeho vstupní fronty již svou kapacitou nestačí, musí přepojovací uzel nově přijímané bloky dat zahazovat, protože je nemá kam uložit. Mechanismy potvrzování, které se snaží zajistit spolehlivý přenos, reagují na obě tyto situace opakovaným vysíláním již jednou vyslaných dat, tedy vlastně dalším zvýšením provozu v síti, což dále zhoršuje přetížení přepojovacích uzlů, a může vést až k úplnému zhroucení sítě v důsledku zahlcení (**congestion collapse**).

Jednou z možností, jak předcházet zahlcování přepojovacích uzlů, je dát jim možnost upozornit odesilatele na hrozící nebezpečí. To však opět stojí určitou část přenosové kapacity. Další, méně nákladnou možností, je vhodná disciplína samotných odesilatelů.

Protokol TCP se snaží předcházet zahlcování přepojovacích uzlů tím, že odesilatel si sám zmenšuje šířku svého datového okénka v situaci, kdy předpokládá hrozící zahlcení přepojovacích uzlů. Každou ztrátu datového segmentu (tj. každý případ, kdy nedostane kladné potvrzení do příslušného časového limitu), interpretuje odesilatel jako důsledek zahlcení některého z přepojovacích uzlů na cestě k příjemci, a reaguje na to tím, že si sám zúží své datové okénko na polovinu (současně s tím prodlužuje na

dvojnásobek i časový limit, do kterého očekává kladné potvrzení, viz minule). Při opakovaných ztrátách přenášených segmentů tak velikost okénka exponenciálně klesá, čímž se odesilatel snaží maximálně snížit objem provozu v síti, a poskytnout tak přepojovacím uzlům potřebný čas na zotavení. Tato technika je označována jako **Multiplicative Decrease Congestion Avoidance.**

Otázkou ovšem je, jak má protokol TCP reagovat na konec stavu zahlcení. Kdyby začal opět exponenciálně zvětšovat velikost svého datového okénka, vedlo by to na prudkou oscilaci celé přenosové sítě mezi stavem s minimálním objemem přenosů a stavem zahlcení. Proto musí protokol TCP postupovat "umírněněji", a své datové okénko zvětšovat pomaleji (způsobem, který je označován jako **slow-start recovery**, doslova: zotavení s pomalým startem). Za každý úspěšně přenesený segment si odesilatel zvětší své datové okénko jen o velikost tohoto segmentu. Jakmile však dosáhne poloviny té velikosti okénka, kterou mu nabízí příjemce (viz výše), postupuje ještě pomaleji, a velikost okénka zvětšuje vždy až poté, co dostane kladné potvrzení o úspěšném přenosu všech segmentů v rámci celého okénka. Maximální velikostí okénka je pak samozřejmě ta, kterou mu signalizuje příjemce.

Kombinací všech výše uvedených mechanismů pro řízení toku, spolu s adaptivním způsobem potvrzování (o kterém jsme si povídali minule) dosahuje protokol TCP velmi efektivního využití existujících přenosových kapacit. Praktické testy ukázaly, že oproti prvním verzím protokolu TCP, které uvedené mechanismy ještě nepoužívaly, dosahují novější verze TCP dvakrát až desetkrát většího přenosového výkonu. To je jistě jeden z hlavních důvodů dnešní velké popularity protokolu TCP.

60/ Protokol TCP - IV.

Po předcházejících třech dílech, ve kterých jsme se zabývali celkovou filosofií protokolu TCP a jeho přístupem k řešení různých problémů, již můžeme být poněkud konkrétnější. Můžeme si ukázat přesný formát datových bloků, se kterým protokol TCP pracuje.

Nejprve si ale připomeňme (v souladu s 57. dílem), že bloky dat, přenášené protokolem TCP jako celek, se označují jako **segmenty** (**segments**), zatímco v případě alternativního protokolu UDP jde o tzv. **uživatelské datagramy** (**user datagrams**). V obou případech je přenášený datový blok tvořen dvěma hlavními částmi: hlavičkou (header) a vlastními daty. Zatímco ale v případě protokolu UDP stačí, když hlavička kromě zabezpečovacího údaje a příznaku délky obsahuje již jen údaj o portu koncového příjemce a odesilatele (viz obr. 56.2 v 56. dílu), hlavička TCP segmentu musí být obsažnější - jak je ostatně možné očekávat v případě výrazně složitějšího přenosového protokolu. Přesný formát segmentu protokolu TCP ukazuje obrázek 60.1.

.PI OBR60_1.TIF, 20, 40, 10

Obr. 60.1.: Formát TCP segmentu

Čím je určeno spojení

Spojované (connection-oriented) protokoly, mezi které patří i protokol TCP, musí vždy nějakým způsobem identifikovat konkrétní spojení, po kterém jsou určitá data

přenášena. Spojované protokoly síťové vrstvy mají blíže k představě spojení jako kanálu, který někudy vede, a tak jej obvykle označují jednoznačným identifikátorem (např. číslem). Tento identifikátor pak používají jak pro zanesení příslušné cesty do směrovacích tabulek mezilehlých uzlů, přes které spojení prochází, tak i ve vlastních datových paketech pro určení cesty, kterou mají být přenášeny (místo adresy příjemce). Spojované protokoly na úrovni transportní vrstvy (což je případ protokolu TCP) však již "nevidí" žádné mezilehlé uzly, nepotřebují brát je v úvahu, a je pro ně přirozenější identifikovat spojení dvojicí: <*počáteční bod, koncový bod*>.

Co je ale koncovým a počátečním bodem spojení v případě protokolu TCP? IP adresa příjemce a odesilatele nestačí, neboť ta jednoznačně identifikuje pouze uzlový počítač, a nikoli již konkrétní entitu aplikační vrstvy v rámci daného uzlu. Obdobně není samo o sobě dostatečné ani číslo portu (viz 55. díl), které zase identifikuje aplikační entitu v rámci daného počítače. Počáteční (a stejně tak i koncový) bod spojení může být určen až dvojicí <*IP adresa, číslo portu*>.

.PI OBR60_2.TIF, 20, 40, 10

Obr. 60.2.: TCP segment a jeho pseudohlavička

Jak ovšem vyplývá z obrázku 60.1., v TCP segmentu jsou vyjádřeny pouze čísla portů odesilatele a příjemce. Potřebné IP adresy jsou pak, obdobně jako v případě protokolu UDP, součástí tzv. pseudohlavičky (viz obr. 60.2), která je shodná s pseudohlavičkou protokolu UDP (samozřejmě až na obsah pole **PROTO**, které identifikuje transportní protokol, a pro TCP má hodnotu 6). Stejný jako u protokolu UDP je i význam pseudohlavičky a způsob jejího využití - je zahrnuta do výpočtu kontrolního součtu (položka CHECKSUM v TCP segmentu), ale ve skutečnosti přenášena není. Obsah jednotlivých položek (především pak obě IP adresy) dostává transportní vrstva na straně příjemce od vrstvy síťové - stejným způsobem, který jsme si popisovali již v 56. dílu.

Kladné potvrzování

Z předchozích dílů již také víme, že protokol TCP používá kladné potvrzování - nikoli ovšem celých bloků (segmentů), ale jednotlivých bytů (osmibitových oktetů). Příjemce i odesilatel se dívají na přenášená data jako na lineární posloupnost bytů, a v každém segmentu se vlastně přenáší část této lineární posloupnosti. Která část to je, udává položka SEQUENCE NUMBER v hlavičce segmentu (vyjadřující pozici prvního přenášeného bytu v této posloupnosti). Kladné potvrzení, které příjemce vrací odesilateli v případě úspěšného příjmu, pak má opět formu pozice bytu v uvažované lineární posloupnosti - tentokráte ale pozice prvního bytu, který příjemce očekává jako další (položka ACKNOWLEDGEMENT NUMBER). V položce WINDOW je pak vyjádřena velikost datového okénka, které příjemce nabízí odesilateli, a které ve své podstatě signalizuje, kolik volného místa ve své vyrovnávací paměti má příjemce ještě k dispozici (viz minulý díl).

Je dobré si uvědomit, ke kterému směru přenosu se obsah tří právě uvedených položek vztahuje. Uvažujeme-li segment, směřující od uzlu A do uzlu B, týká se jeho položka SEQUENCE NUMBER přenosu v témže směru (tj. od A do B), zatímco potvrzení v položce ACKNOWLEDGEMENT NUMBER se vztahuje k přenosu dat v opačném směru (tj. od B do A). Stejně tak se i obsah položky WINDOW týká přenosu dat v opačném směru (od B do A).

Naléhavá data

Jak jsme si již také naznačili, protokol TCP nabízí své přenosové služby ve formě proudu (stream) jednotlivých bytů - odesilatel na jedné straně zadává k odeslání postupně jednotlivé byty, a příjemce si je na druhé straně ve stejném pořadí vyzvedává. Lze si tedy představit, že protokol TCP implementuje pomyslnou rouru, do které odesilatel z jedné strany jednotlivé byty vkládá, a příjemce si je z druhé strany zase odebírá. Uvnitř této roury se však může nacházet nezanedbatelný počet bytů, které již byly odeslány, ale nebyly dosud přijaty. A to může u některých aplikací způsobovat nemalé problémy. Například při vzdáleném přístupu pomocí terminálové emulace se uvnitř pomyslné roury nachází znaky, které uživatel již zadal z klávesnice, ale které ještě nebyly hostitelským počítačem přijaty a zpracovány. Potřebuje-li však uživatel náhle zadat vhodný povel k přerušení právě probíhající činnosti (tzv. break), musely by příslušné řídící znaky postupně projít rourou a mohly by se uplatnit až teprve tehdy, když budou zpracovány všechny předtím zadané znaky.

Protokol TCP však právě pro tyto účely nabízí možnost "předbíhání", dovolující vyslat příslušné povely jako tzv. **naléhavá data** (**urgent data**), někdy označovaná též jako **data mimo pořadí** (**out-of-band data**). Příznak toho, že daný segment obsahuje naléhavá data, je vyjádřen nastavením příslušného bitu v položce CODE BITS. Vlastní naléhavá data pak musí být umístěna od začátku datové části segmentu, a v položce URGENT POINTER je ukazatel na jejich konec v rámci daného segmentu.

Po doručení naléhavých dat by měl být koncový příjemce neprodleně upozorněn na jejich existenci. Samotný protokol TCP však již nestanovuje, jakým způsobem se tak má dít. Ostatně ani dost dobře nemůže, neboť příslušný mechanismus závisí na konkrétní implementaci.

Další řídící bity v šestibitové položce CODE BITS pak slouží mj. potřebám navazování a rušení spojení, operaci push (viz 57. díl), umožňují zneplatnit položku ACKNOWLEDGEMENT NUMBER atd.

Segment ani jeho hlavička nemají pevnou délku

Velikost datové části segmentu, a tím i segmentu jako takového, není pevná, a stejně tak nemusí být stejně velké ani segmenty, přenášené postupně v rámci téhož spojení. Celková délka segmentu (tj. jeho datové části i hlavičky) přitom není vyjádřena přímo v segmentu samotném, ale až v jeho pseudohlavičce (viz obr. 60.2., položka TCP LENGTH).

Otázkou ovšem je, jak volit optimální velikost segmentu. Zde je nutné si uvědomit, že jednotlivé segmenty jsou vkládány do IP datagramů, a ty zase do rámců (na úrovni rámců vrstvy síťového rozhraní) - každý z nich si ale k "užitečným" datům přidává svá řídící data, která představují režii přenosu. V případě malých segmentů je tato režie relativně velká, zatímco v případě velkých segmentů dochází k fragmentaci, když nelze umístit celé segmenty resp. IP datagramy do přenosových rámců. Optimální by byl co největší segment, při kterém ještě nebude docházet k žádné fragmentaci. Najít jeho velikost je ale velmi netriviální (tato velikost totiž závisí na mnoha faktorech, které se navíc mohou dynamicky měnit). Samotný protokol TCP neobsahuje žádný mechanismus pro nalezení optimální velikosti segmentu.

Co však protokol TCP nabízí, je prostředek, pomocí kterého se obě strany mohou dohodnout alespoň na maximální velikosti přenášených segmentů. Implicitně používá

TCP maximum, které odpovídá datové části segmentu o velikosti 536 bytů (tak, aby se segment včetně své obvyklé hlavičky vešel do IP datagramu implicitní velikosti 576 bytů). Tato implicitní hodnota je založena spíše na pesimistickém předpokladu, že spojení mezi oběma koncovými účastníky není přímé, ale vede přes jednu či několik bran, a má malou propustnost. Pokud jsou však oba koncoví účastníci připojeni například k témuž segmentu rychlé lokální sítě, je jistě žádoucí, aby plně využívali její větší šířku přenosového pásma a její schopnost přenášet najednou větší bloky dat. Tedy volit větší maximální velikost segmentů.

Konkrétní mechanismus pro stanovení maximální velikosti segmentu využívá volitelné položky OPTIONS v hlavičce segmentu (viz obr. 60.1.). Je-li ovšem tato položka volitelná, stejně tak jako je proměnná její délka, nemůže mít pevnou délku ani samotná hlavička TCP segmentu. Konkrétní velikost hlavičky (rozšířené výplní ze samých nul na násobek 16 bitů) je pak vyjádřena v položce HLEN.

61/ Transportní rozhraní

V minulých dílech našeho seriálu jsme se podrobněji zabývali síťovou a transportní vrstvou síťového modelu TCP/IP. Dříve, než se budeme věnovat aplikační vrstvě a jednotlivým aplikacím, se musíme ještě zmínit o jedné důležité věci: o vazbě mezi transportní vrstvou a aplikačními programy. Tedy o tom, jaké mechanismy a jaké postupy mohou, resp. musí aplikační programy používat, chtějí-li získat přístup ke službám transportní vrstvy.

Ještě dříve si ale musíme uvědomit jednu významnou skutečnost: implementace nejnižších vrstev určitého síťového modelu je vždy součástí operačního systému počítače, zatímco nejvyšší vrstva (aplikační) již součástí operačního systému není. O přesné hranici, tedy o tom, která vrstva ještě je součástí operačního systému a která již ne, pak rozhoduje především samotný síťový model a jeho představa o tom, jak má být síťový software členěn na vrstvy. V případě síťového modelu TCP/IP je za součást operačního systému považována ještě transportní vrstva, zatímco vrstva aplikační již stojí nad operačním systémem.

To má ale jeden velmi závažný důsledek: přístup ke službám transportní vrstvy TCP/IP zprostředkovává aplikacím operační systém. Způsob, jakým se tak děje - tedy jednotlivé mechanismy a postupy - jsou pak dány konkrétním operačním systémem a jeho celkovou koncepcí. Proto je dobré rozlišit dvě věci: protokoly, používané na úrovni transportní vrstvy, a rozhraní transportní vrstvy směrem k vrstvě aplikační (tzv. transportní rozhraní). Transportní protokoly musí být na všech vzájemně komunikujících počítačích stejné, a proto mohou být v rámci síťového modelu TCP/IP standardizovány. Transportní rozhraní je ale závislé na konkrétním operačním systému, a na různých počítačích tedy může být různé. Proto dost dobře nemůže být v rámci síťového modelu TCP/IP jakkoli standardizováno.

V této souvislosti je vhodné si znovu zdůraznit, že TCP/IP rozhodně není to samé, co Unix. Unix je jeden z mnoha operačních systémů, a TCP/IP je jedna z mnoha soustav protokolů a názorů na to, jak mají vypadat a fungovat počítačové sítě (tedy to, co zde označujeme jako síťový model). Tvůrci operačního systému Unix si síťový model TCP/IP oblíbili natolik, že pro potřeby propojování svých počítačů do sítí již nepoužívají nic jiného. To ale rozhodně neznamená, že by v prostředí Unixu nemohly být implementovány jiné síťové modely resp. architektury. Stejně tak to rozhodně

neznamená, že by síťový model TCP/IP nemohl být implementován i v prostředí jiných operačních systémů. Dnes také skutečně existují implementace TCP/IP snad pro každý existující operační systém, od MS DOSu pro počítače PC až po operační systém VM/CMS střediskových počítačů firmy IBM. Je zde však přeci jen určitá odlišnost, která svádí ke ztotožňování TCP/IP s Unixem: implementace protokolů TCP/IP je dnes již považovaná za standardní součást Unixu, tj. je přímou součástí dodávky tohoto operačního systému. V případě ostatních operačních systémů je však pouze jedním z mnoha doplňkových produktů, které si uživatel dokupuje zvlášť (a často si také může vybrat z širší nabídky různých implementací).

Jestliže tedy mohou být protokoly TCP/IP implementovány v prostředí různých operačních systémů, znamená to současně, že v každém z nich budou služby transportní vrstvy přístupné jiným způsobem. To je velmi důležité nejen pro samotné aplikace a jejich tvůrce, a také pro způsob, jakým mohou být v rámci síťového modelu TCP/IP standardizovány nejobvyklejší aplikace typu elektronické pošty, přenosu souborů, vzdálených terminálových relací apod. Standardizovat je možné (a nutné) celkovou koncepci těchto aplikací a konkrétní protokoly, prostřednictvím kterých budou spolupracovat (např. předávat si jednotlivé zprávy v rámci elektronické pošty, celé soubory v rámci aplikací pro přenos souborů atd.). Není ale již možné standardizovat přesný způsob, jakým budou tyto aplikace využívat služeb transportní vrstvy.

Budeme-li si dále popisovat konkrétní řešení transportního rozhraní v prostředí Unixu, je dobré mít na paměti, že jde jen o konkrétní příklady možného řešení.

Jak je to v Unixu

Na vznik Unixu měl významný vliv operační systém Multics, který byl vyvíjen v rámci velkého projektu několika významných institucí (MIT, General Electric a Bell Laboratories) jako operační systém pro práci v režimu sdílení času. Celý projekt v podstatě ztroskotal, protože výsledný operační systém postupným zdokonalováním tak nabubřel a zkomplikoval se, že se stal prakticky nepoužitelným. Spíše negativní zkušenosti některých účastníků tohoto projektu se pak staly jednou z hlavních motivací pro vznik Unixu.

Pro nás je zajímavé to, že operační systém Multics jako první sjednotil způsob práce se vstupně/výstupními zařízení - na všechna tato zařízení se díval jako na zvláštní soubory, a pracoval s nimi stylem "otevři - čti - piš - zavři".

Když pak Bellovy laboratoře (Bell Labs, součást tehdy ještě mamutího AT&T) odstoupily od projektu Multics, vznikl na jejich půdě (v roce 1969) operační systém Unix. Jeho autoři (Ken Thompson a jeho spolupracovník Dennis Ritchie) do něj převzali i jednotný pohled na vstupně/výstupní zařízení z původního Multicsu. Unix se tedy již od svého vzniku snažil dívat se na všechna vstupně/výstupní zařízení jako na speciální soubory, a jako s takovými s nimi také pracoval. Na úrovni operačního systému proto nabízel operace typu: *open* (otevření souboru), *read* (čtení z již otevřeného souboru), *write* (zápis do souboru) a *close* (uzavření dříve otevřeného souboru, signalizující konec práce s ním), pomocí kterých se pak realizovaly veškeré vstupně/výstupní operace. Speciálním souborem, se kterým se přitom pracovalo, byl ve skutečnosti ovladač příslušného zařízení (který se jako soubor pouze "tvářil").

Tento pohled na vstupně/výstupní zařízení a způsob práce s nimi si Unix v podstatě ponechal dodnes, a pro běžná vstupně/výstupní zařízení (terminály, disky apod.) je toto řešení vcelku postačující.

Když se pak v prostředí Unixu začaly implementovat síťové protokoly TCP/IP, bylo vcelku přirozené, že se nejprve použil stejný mechanismus. Síť byla považována za vstupně/výstupní zařízení, a "tvářila" se jako speciální soubor. Příslušné protokoly pak byly implementovány v rámci ovladače příslušného zařízení, který navenek vystupoval právě jako speciální soubor.

Záhy se ale ukázalo, že způsob práce se sítí je přeci jen komplikovanější a náročnější, než způsob práce se skutečnými vstupně/výstupními zařízeními, a že tudíž vyžaduje jiné, obecnější mechanismy. Ukažme si to na příkladu.

Operace *open*, kterou se v Unixu otevírá soubor, signalizuje operačnímu systému, že daná úloha (proces) chce pracovat s určitým konkrétním souborem - jeho přesná specifikace musí být při volání operace *open* zadána. V případě, že je operace *open* provedena nad speciálním souborem, který implementuje příslušné přenosové protokoly, by analogií k otevření skutečného souboru mělo být navázání spojení na úrovni transportní vrstvy, tak aby následné operace *read* a *write* již mohly přenášet "užitečná data", a operace *close* zase mohla spojení zrušit. To by ovšem také znamenalo zadat v okamžiku navazování spojení (volání operace *open*), s kým má být spojení navázáno (místo specifikace souboru, který má být otevřen). Problém je ale v tom, že některé aplikace nemusí být v okamžiku navazování spojení vůbec schopny určit, s kým má být navázáno (jde-li například o proces, který plní funkci serveru, a bude teprve čekat na to, až mu nějaký klient zavolá). Místo jedné operace *open*, zajišťující navazování spojení, by bylo zapotřebí rozlišovat operace dvě: tzv. pasivní otevření, které pouze signalizuje operačnímu systému připravenost k přijetí žádosti (přicházející z druhé strany) o navázání spojení, a tzv. aktivní otevření, které spojení skutečně navazuje. To vše by ale platilo zase jen pro spojované transportní služby, realizované protokolem TCP, zatímco pro nespojované transportní služby protokolu UDP (kde se spojení nenavazuje vůbec) by operace *open* musela fungovat ještě jiným způsobem.

Specifické požadavky práce se sítí by samozřejmě bylo možné řešit úpravou stávajících operací typu *open, read, write* a *close*, šitých na míru práci se soubory - například zavedením dalších parametrů. Co by ale například znamenalo pasivní a aktivní otevření souboru? Musela by se při otevírání skutečného souboru uvádět také IP adresa, která je doopravdy potřebná jen při zřizování transportního spojení? Pokud by jedna a tatáž operace (například *open*) měla v různých kontextech různé vstupní parametry, zase by se tím ztratila výhoda jednotného pohledu na všechna zařízení jako na soubory.

Zavedení obecnějších mechanismů, které by lépe vyhovovaly potřebám přenosových protokolů a umožnily vytvořit dostatečně pružné transportní rozhraní, se tak brzy stalo nezbytností.

Dvě větve Unixu

V době, kdy potřeba obecnějších mechanismů vyvstala, byl již vývoj operačního systému Unix rozdělen do dvou hlavních větví. Jednou z nich byl tzv. BSD Unix (či Berkeley Unix), vyvíjený ve středisku Berkeley Software Distribution na University

of California v Berkeley, zatímco druhou větev představoval Unix, vyvíjený firmou AT&T (a v současné době označovaný jako System V).

Jak jsme si již uvedli ve 42. dílu našeho seriálu, byly protokoly TCP/IP poprvé implementovány právě pro BSD Unix, a to firmou BBN (Bolt, Beranek & Newman) na zakázku od vládní agentury DARPA, která také financovala vývoj protokolů TCP jako takových. Pro BSD Unix pak byl vytvořen nový mechanismus na úrovni operačního systému, který aplikačním programům zprostředkovává přístup ke komunikačním protokolům. Je označován jako **socket interface** (doslova: rozhraní s objímkami, zásuvkami), a příště se mu budeme věnovat podrobněji.

Do AT&T Unixu se protokoly TCP/IP prosadily až o něco později, a pro jejich začlenění do operačního systému byl použit jiný mechanismus, označovaný jako **streams** (doslova: proudy, toky). Také o něm si povíme podrobněji.

62/ Transportní rozhraní - BSD Sockets

V minulém dílu jsme dospěli k tomu, že implementace protokolů transportní vrstvy, stejně tak jako všech nižších vrstev, je součástí operačního systému, a služby těchto vrstev jsou jednotlivým aplikacím poskytovány pomocí takových mechanismů, které jsou v daném operačním systému k dispozici - a které samozřejmě mohou být v různých operačních systémech různé. Pak jsme se již zaměřili na prostředí operačního systému Unix a řekli si, že ten se zpočátku snažil uplatnit i na síť svůj jednotný pohled na všechna zařízení, která chápe jako speciální soubory, a jako se soubory s nimi také pracuje. Řekli jsme si však také, že právě v případě síťových služeb se původní prostředky pro práci se soubory záhy ukázaly být nedostačující, a proto musely být vyvinuty nové, obecnější mechanismy. Závěrem jsme si pak také naznačili, že v každé z obou hlavních větví Unixu byly za tímto účelem vytvořeny jiné mechanismy. Dnes si podrobněji ukážeme, jak je tomu v tzv. BSD Unixu.

Lidé, kteří dostali za úkol vyvinout nové mechanismy pro zpřístupnění protokolů TCP/IP v prostředí BSD Unixu, si svůj původní úkol poněkud rozšířili: rozhodli se vytvořit takový prostředek, který by nebyl "šit na míru" jedné konkrétní soustavě protokolů, tj. TCP/IP, ale byl použitelný obecně pro jakoukoli soustavu protokolů, a dokonce i pro více soustav protokolů současně. V době, kdy si protokoly TCP/IP v prostředí Unixu teprve hledaly své místo, to bylo rozhodnutí vcelku logické. V současné době, kdy TCP/IP protokoly Unix zcela ovládly, není tato možnost příliš využívána. Do budoucna by se ale zmíněné rozhodnutí mohlo ukázat jako velmi prozíravé. Pokud se totiž referenční model ISO/OSI opravdu prosadí do života, nebo se alespoň stane skutečnou konkurencí pro síťový model TCP/IP, bude začlenění ISO protokolů do BSD Unixu velmi jednoduché.

Větší obecnost nově vytvořeného mechanismu, umožňující zpřístupnit aplikacím přenosové služby různých soustav protokolů, však není zcela zadarmo. Cenou za tuto větší obecnost je samozřejmě poněkud větší komplikovanost použití. Kdyby šlo o mechanismus, určený pouze pro protokoly TCP/IP, mohl by například vždy sám předpokládat, že kdykoli se mu zadá nějaká adresa, jde o tzv. IP adresu (viz 44. díl seriálu). Takto je nutné mu to explicitně sdělit.

Co je vlastně socket?

Nový mechanismus pro zpřístupnění služeb transportních protokolů, vytvořený pro tzv. BSD Unix (tj. pro Unix, vyvíjený ve středisku Berkeley Software Distribution při University of Berkeley), je označován jako **socket interface**, nebo zkráceně jen jako **sockets** (doslova: objímky, zásuvky. V prvním přiblížení lze říci, že jde o zobecnění původního mechanismu Unixu pro práci se soubory, a že každý "socket" je vlastně koncovým bodem komunikace.

Abychom si ale mohli ukázat, v čem toto zobecnění spočívá, musíme se ještě vrátit k souborům a způsobu práce s nimi.

Když se nějaký proces rozhodne pracovat s určitým souborem, musí jej nejprve otevřít. Vlastní otevření však zajišťuje operační systém, a proto si provedení potřebné operace (operace *open*) proces vyžádá na operačním systému. Konkrétní forma je taková, že proces použije tzv. systémové volání (system call), neboli spustí podprogram, který je součástí operačního systému. Svůj konkrétní požadavek (který soubor má být otevřen, jakým způsobem atd.) proces specifikuje prostřednictvím parametrů tohoto volání. Vlastní otevření souboru pak probíhá plně v režii operačního systému, který si za tímto účelem založí vhodnou datovou strukturu, a v ní si udržuje vše, co k práci se souborem potřebuje.

Proces, který si otevření souboru vyžádal, pak ještě musí mít k dispozici prostředek, pomocí kterého při všech následných operacích nad již otevřeným souborem (tj. operacích *read, write* a *close* atd.) určí, kterého souboru se požadované operace týkají. Nejlépe by se k tomuto účelu hodila zmíněná datová struktura, ale ta je zcela ve správě operačního systému, který ji aplikačním procesům nezveřejňuje. Přesto ale umožňuje aplikačním procesům odkazovat se na tuto strukturu, a to prostřednictvím nepřímého ukazatele (který je ve skutečnosti celým číslem bez znaménka, a lze si jej představit jako index do tabulky, obsahující ukazatele na zmíněné datové struktury). Operace *open* proto vrací jako svůj výsledek celé číslo (tzv. deskriptor souboru, *file descriptor),* a ten pak používají operace *read, write* a *close* k určení souboru, ze kterého se má číst, do kterého se má zapisovat, resp. který má být uzavřen.

Nyní si již můžeme snadno naznačit, co je podstatou nového mechanismu "socket interface". Samotný socket není nic jiného, než jiná varianta výše popsané datové struktury, ve které si operační systém uchovává různé údaje (tentokráte pro potřeby komunikace v síti). Socket opět není přímo přístupný, ale aplikační procesy se na něj mohou odkazovat přesně stejným způsobem, jako na zmíněnou datovou strukturu při práci se soubory - tedy nepřímo, prostřednictvím celého čísla. To se nyní označuje jako deskriptor socketu (*socket descriptor*), ale svým datovým typem je totožné s deskriptorem souboru (jde o celé číslo bez znaménka).

Stejný je také způsob práce se sockety - prostřednictvím systémových volání. Samotný repertoár systémových volání pro práci se sockety je však samozřejmě jiný, než pro práci se soubory. BSD Unix se však snaží zachovávat maximální možnou slučitelnost obou mechanismů, tak aby se lišily jen tam, kde je to skutečně nutné. Proto například umožňuje používat operace *read* a *write* jak pro čtení a zápis dat do souborů, tak i pro jejich vlastní přenos po síti prostřednictvím socketů. Za tímto účelem mj. vybírá celá čísla, která přiděluje jako deskriptory, z jediné množiny. Nemůže se tudíž stát, aby se číselná hodnota deskriptoru souboru a deskriptoru socketu rovnala.

Vytváření socketů

Společnou vlastností obou mechanismů je skutečnost, že příslušné datové struktury vznikají až v okamžiku jejich skutečné potřeby - při otevírání souboru, resp. při řádosti o vytvoření socketu, a vytváří je operační systém na základě bezprostředního požadavku aplikačního procesu. Významný rozdíl je ale v tom, co všechny musí být v okamžiku vytváření příslušných datových struktur zadáno: v případě otevírání souborů to musí být explicitní údaj o konkrétním souboru, který má být otevřen. Naproti tomu při zřizování socketu se ještě neuvádí, s kým se bude jeho prostřednictvím komunikovat.

Tím se vychází vstříc potřebám nejrůznějších procesů, které vystupují v roli serverů, potřebují si nechat vytvořit socket pro komunikaci se svými klienty, ale teprve následně se dozvídají, kteří klienti s nimi budou chtít komunikovat.

Žádost o vytvoření nového socketu, kterou aplikační proces předává operačnímu systému formou systémového volání (s příznačným názvem *socket*) však ve svých parametrech obsahuje údaj o tom, s jakou soustavou protokolů bude socket pracovat (což mj. definuje význam následně zadávaných adres), dále druh spojení (spolehlivé spojované či nespolehlivé nespojované, případně spojení na nižší úrovni, než je transportní), a také konkrétní protokol z příslušné rodiny protokolů, který bude přenos zajišťovat (což pamatuje na případ, kdy požadovaný druh spojení může být v rámci zvolené soustavy protokolů realizován více alternativními protokoly).

Vazba socketu na lokální port

Jestliže jsme si již dříve uvedli, že na socket se můžeme dívat jako na koncový bod komunikace na daném uzlovém počítači, jaký je v tomto případě jeho vztah k portům? Jak jsme si uvedli v 55. dílu, je právě port koncovým bodem komunikace na rozhraní mezi transportní vrstvou a vrstvou aplikační.

Zde je ovšem třeba si uvědomit, že porty jsou jednotným konstruktem v rámci celého síťového modelu TCP/IP, zatímco sockety jsou jen jedním z možných prostředků pro zpřístupnění transportních služeb aplikačním procesům (tím, který je používán v BSD Unixu). Jednotnost portů ve všech uzlech umožňuje jednotné adresování entit aplikační vrstvy, bez ohledu na to, jaký konkrétní mechanismus používají tyto entity pro přístup ke službám transportní vrstvy. Jsou-li tímto mechanismem sockety, existuje mezi nimi a porty jednoznačná korespondence: každý socket odpovídá jednomu portu, resp. je svázán s právě jedním portem.

Tato vazba mezi portem a socketem však není automatická. Přesněji: nevzniká při vytvoření socketu (systémovým voláním *socket*), ale musí být vytvořena až následně, na explicitní příkaz (systémovým voláním *bind*).

Volba lokálního portu, na který má být socket navázán, může být pro některé aoplikační procesy irrelevantní, a tyto procesy pak mohou nechat volbu konkrétního portu na operačním systému. Jinak je tomu ale u procesů, které vystupují v roli serverů, a své služby poskytují na tzv. dobře známých portech (se kterými předem počítají potenciální klienti těchto služeb), viz 55. díl seriálu.

Vazba socketu na vzdálený port

Vytvoření vazby socketu na některý z lokálních portů na daném uzlovém počítači stále ještě nic neříká o tom, s kým se bude prostřednictvím daného socketu

komunikovat - tj. s kterým jiným uzlovým počítačem a kterou aplikační entitou na tomto počítači. Za tímto účelem je BSD Unix vybaven dalším systémovým voláním (*connect*), kterému se jako parametr zadá adresa vzdáleného počítače a číslo portu na tomto počítači. Výsledný efekt systémového volání *connect* pak ovšem závisí na tom, jaký druh spojení byl specifikován při vytváření socketu (pomocí volání *socket).* Pokud šlo o spojení spojovaného charakteru, je v rámci volání *connect* spojení skutečně vytvořeno (tj. navázáno). Pokud jde o komunikaci nespojovaného charakteru, je volání *connect* nepovinné, a má vlastně jen vedlejší efekt v tom, že specifikuje adresu, na kterou budou následně odesílány jednotlivé datagramy - tak aby tato adresa nemusela být s každým datagramem zadávána vždy znovu.

Čekání na žádost o spojení

Aplikační proces, který sám iniciuje přenos dat, si tedy nejprve nechá vytvořit socket (voláním *socket*), pak jej sváže s určitým místním portem (voláním *bind*), a posléze i s portem na vzdáleném počítači (voláním *connect*), jde-li o spojovanou komunikaci. Pak již může přistoupit k vysílání či příjmu vlastních dat. Za tímto účelem má k dispozici hned několik variant operací *read* a *write* (realizovaných opět formou systémových volání), které jsou v principu shodné se čtením a zápisem dat do souborů.

Jak má ale postupovat proces, který sám komunikaci iniciovat nechce? Tedy například proces, který chce vystupovat v roli serveru, a chce přitom používat spojovaný charakter komunikace?

Také si musí nejprve nechat vytvořit potřebný socket (voláním *socket*), a pak jej navázat na vhodný lokální port (voláním *bind*). Pak ale musí uvést socket do takového stavu, který odpovídá pasivnímu otevření - tedy kdy socket (resp. na něj navázaný port) pouze pasivně čeká na přicházející žádosti o navázání spojení. Za tímto účelem je v BSD Unixu k dispozici další systémové volání *listen*, kterému se navíc jako parametr zadává, jak velkou frontu si má vytvořit pro přicházející volání či data.

Když je takto socket připraven přijímat žádosti o navázání spojení, musí začít čekat i samotný aplikační proces. K tomu má k dispozici další systémové volání *accept*.

Jakmile na port, na kterém proces "čeká", přijde nějaká žádost o navázání spojení, operační systém tuto žádost přijme, a vytvoří nový socket, který "naváže" na port volajícího - čímž vlastně naváže spojení mezi serverem a jeho klientem. Přes tento nový socket bude probíhat vlastní komunikace, zatímco na původním socketu budou dále očekávány požadavky na navázání spojení. Údaj o nově vytvořeném socketu přitom operační systém předává jako výstupní parametr volání *accept*. Aplikační proces v roli serveru pak buďto zajistí následnou komunikaci s klientem sám, a po jejím skončení nově vytvořený socket zase nechá zrušit, nebo si ke zpracování požadavku vytvoří samostatný podproces, a sám se věnuje jen čekání na další žádosti (volá *accept*).

Celý postup komunikace spojovaného charakteru ze strany serveru i klienta ukazuje obr. 62.1., včetně zrušení socketu voláním *close.* V případě nespojované komunikace je celý postup poněkud jednodušší - na straně serveru pak odpadají volání *listen* a *accept*, která se týkají navazování spojení, a na straně klienta nepovinné volání *connect*.

.PI OBR62_1.TIF, 20, 40, 10

Obr. 62.1.: Představa práce se sockety při spojované komunikaci

63/ Transportní rozhraní - Streams a TLI

V minulých dílech seriálu jsme se začali zabývat tím, jak je v prostředí operačního systému Unix implementováno transportní rozhraní k přenosovým protokolům TCP/IP, neboli rozhraní mezi službami transportní vrstvy a jednotlivými aplikacemi. Řekli jsme si, že protokoly TCP/IP se z historických důvodů prosadily nejprve v tzv. BSD větvi Unixu, kde byl pro jejich zpřístupnění vytvořen nový mechanismus - sockets. Dnes je na řadě druhá hlavní větev Unixu, vyvíjená a šířená firmou AT&T (nejprve tzv. Bellovými laboratořemi firmy AT&T, dnes specializovanou dceřinnou organizací USL, neboli Unix System Laboratories).

Pro správné pochopení myšlenek, na kterých je založena implementace protokolů TCP/IP v prostředí AT&T Unixu, je vhodné si nejprve upřesnit časový sled jednotlivých událostí. Mechanismus "sockets", kterým jsme se zabývali minule, byl vyvinut pro verzi 4 BSD Unixu, která se poprvé objevuje v roce 1981. Do AT&T Unixu se obdobný mechanismus dostává až ve verzi System V Release 3 (zkráceně SVR3), kterou firma AT&T ohlásila v roce 1987. Rozdíl šesti let je hodně dlouhou dobou, za kterou se zcela zákonitě musely projevit různé vývojové trendy, nové koncepce i nové požadavky na operační systém Unix jako takový.

Jedním z těchto požadavků byl tlak na větší modularitu a flexibilitu celého vstupně/výstupního subsystému. Operační systém Unix je totiž koncipován tak, že jeho jádro (kernel) obsahuje základní část, nezávislou na konkrétním hardwaru, a dále pak různé ovladače (drivers), které základní části jádra zprostředkovávají vše, co již je závislé na konkrétním hardwaru daného počítače. Tyto ovladače se sice navenek "tváří" jako speciální soubory (a jako s takovými se s nimi také pracuje), ve skutečnosti jsou ale součástí jádra. To však mj. znamená, že při přidání nového ovladače či jakékoli jiné změně je třeba minimálně znovu sestavit celé jádro.

Kromě zřejmé nepružnosti má tato strategie i další nevýhodu, která se nejvíce projevila právě u implementace síťových protokolů. Souvisí s tím, že takto řešené ovladače často znovu implementují funkce, které jsou současně implementovány i v jiných ovladačích. Celková koncepce operačního systému a jeho jádra přitom nevychází příliš vstříc možnosti realizovat tyto společné funkce jen jednou, a příslušné moduly sdílet. Myšlenka řešit síťový software formou ovladačů (drivers), tedy "v jednom balíku", také není příliš v souladu s vrstevnatou koncepcí síťového softwaru, která si přímo říká o modulární implementaci.

Počátkem osmdesátých let, kdy byly protokoly TCP/IP začleňovány do BSD Unixu, nebyl ještě tlak na modularitu a flexibilitu vstupně/výstupního subsystému Unixu takový, aby si vynutil i nový způsob implementace samotných síťových protokolů. V BSD Unixu jsou tyto protokoly stále chápány a implementovány jako ovladače (ovladače speciálních zařízení), a nově je řešen pouze způsob práce s nimi. "Sockets" jsou z tohoto pohledu jen tenkou "slupkou", která ovladače překrývá, a zajišťuje takový způsob přístupu k nim, jaký vyhovuje i nejrůznějším síťovým aplikacím.

Do AT&T Unixu se protokoly TCP/IP dostávají již v době, kdy si nepružnost dosavadní koncepce V/V subsystému sama vynutila změnu. Zhruba v roce 1984 přichází jeden ze spoluautorů Unixu, Dennis Ritchie, s myšlenkou nového mechanismu, příznačně nazvaného **streams** (doslova: **proudy**). Ten pak byl také použit i pro implementaci TCP/IP protokolů v prostředí AT&T Unixu, a pro zpřístupnění těchto protokolů byla vytvořena nová překrývající "slupka", označovaná jako **TLI** neboli **Transport Layer Interface** (rozhraní transportní vrstvy).

Co jsou proudy?

Základní myšlenka proudů (streams) je taková, že mezi ovladač zařízení a proces, který s tímto ovladačem komunikuje, se dynamicky vkládají různé programové moduly (bez nutnosti znovu sestavovat jádro při každém vložení či vyjmutí modulu), a veškerá komunikace mezi procesem a ovladačem pak prochází "skrz" tyto moduly. Představu proudu ilustruje obr. 63.1. b/.

.PI OBR63_1.TIF, 20, 40, 10

Obr. 63.1.: Představa přístupu k ovladači

a/ bez proudů b/ s proudy

Dynamickým vkládáním jednotlivých modulů do proudu vzniká celý řetězec, na jehož jednom konci stojí ovladač (označovaný také jako **stream end**, neboli koncový modul proudu), a na druhém konci tzv. hlavička proudu (**stream head**). Ta zajišťuje potřebné přizpůsobení mezi procesem, který s proudem pracuje, a proudem jako takovým. Mezi hlavičku a ovladač jsou zařazovány moduly, které mohou provádět v podstatě jakékoli akce s daty, které skrz ně prochází. Celý proud přitom pracuje v plně duplexním režimu - data tedy mohou procházet proudem oběma směry současně. Konkrétní způsob komunikace jednotlivých modulů v rámci proudu je takový, že každý vždy zpracuje data, převzatá od svého bezprostředního souseda z jedné strany, a výsledek zpracování zase předá svému sousedovi z druhé strany.

Podstatný je ovšem také způsob, jakým proud vzniká a utváří se. Na počátku je každý proud vytvářen jen jako dvouprvkový, a obsahuje tedy jen vlastní ovladač (stream end) a svou hlavičku (stream head). Zařazování jednotlivých modulů "doprostřed" proudu (konkrétně bezprostředně za hlavičku) se pak provádí dynamicky, na základě skutečné potřeby. Slouží k tomu operace, označovaná příznačně jako *push* (vložení nového modulu do proudu, bezprostředně za hlavičku). Stejně tak je možné jednotlivé moduly z proudu dynamicky vyjímat, a to operací *pop* (která vyjme modul bezprostředně za hlavičkou).

Tímto způsobem je možné zařazovat do proudu moduly, zajišťující nejrůznější funkce - od jednoduchých filtrů a převodníků až po mnohem složitější transformace. Pro potřeby síťového softwaru se přímo vnucuje myšlenka realizovat přenosové protokoly jednotlivých vrstev ve formě takovýchto modulů, a pak z nich dynamicky sestavovat celé hierarchické sestavy protokolů (protocol stacks). Tedy například transportní protokoly TCP a UDP realizovat jako dva (alternativní) moduly, síťový protokol IP jako další modul, a na ovladači (např. ovladači síťového adaptéru) ponechat přímé ovládání hardwaru (resp. protokoly, spadající do vrstvy síťového rozhraní).

Transport Layer Interface

Právě popsaný mechanismus proudů (streams) je však stále jen pouhým zdokonalením ovladačů v prostředí AT&T Unixu verze System V. Není určen jen pro implementaci síťových protokolů, ale používá se i k jiným účelům - například k obsluze terminálů apod.

Pro nás je ovšem podstatné, že výsledný proud (stream) se nadále "tváří" jako speciální soubor, a způsob jeho implementace (pomocí proudu) tudíž není pro aplikační procesy příliš relevantní. Pro ně je naopak zapotřebí obdobná "slupka", jakou je BSD Unixu rozhraní socket interface (viz minule), která by překryla příslušné proudy, a zpřístupnila transportní služby takovým způsobem, jaký vyhovuje aplikačním procesům.

V BSD Unixu je zmíněná "slupka" realizována formou systémových volání, což jsou ve skutečnosti vstupní doby do jádra, a teprve to pak zajišťuje veškeré požadované služby. Datové struktury, které se v této souvislosti používají (tj. vlastní sockety) jsou vytvářeny v paměti, která také patří jádru, a to si zabezpečuje vše potřebné pro alokaci a následné uvolňování této paměti.

U AT&T Unixu jsou analogií systémových volání v BSD Unixu volání knihovních rutin, které se přilinkovávají k aplikačním procesům, a jsou tudíž provozovány v adresovám prostoru těchto procesů. Stejně tak jsou v adresovém prostoru aplikačních procesů alokovány i veškeré datové struktury, které jsou těmito knihovními rutinami využívány.

Rozhraní k aplikačním procesům, které tyto knihovní rutiny vytváří, je v AT&T Unixu verze System V označováno jako **Transport Layer Interface** (zkráceně: **TLI,** doslova: **rozhraní transportní vrstvy**).

.PI OBR63_2.TIF, 20, 40, 10

Obr. 63.2.: Sockets (a/) vs. transport endpoints (b/)

Operace, které toto rozhraní nabízí, jsou v mnohém obdobné operacím, které se používají pro práci se sockety v BSD Unixu. Významnějším rozdílem je však povaha objektů na rozhraní mezi transportní a aplikační vrstvou. Zatímco v BSD Unixu je tímto objektem datová struktura (tj. socket), v případě TLI hraje tutéž roli spojení dvou entit - uživatele služby (který sídlí v aplikační vrstvě), a poskytovatele služby (který sídlí ve vrstvě transportní), viz obr. 63.2.b/. Toto lokální spojení je v terminologii TLI označováno jako **transport endpoint** (doslova: koncový bod transportního spojení), a je tedy analogií socketu v BSD Unixu. Zřizuje se pomocí rutiny *t_open,* která je obdobou systémového volání *open* v BSD Unixu, a teprve následně s ním musí být sdruženo i vhodné číslo portu (jako adresa transportního spojení), rutinou *t_bind* (která je analogická systémovému volání *bind* v BSD Unixu).

Další postup na straně serveru ukazuje obrázek 63.3.: po volání rutiny *t_bind* si příslušný aplikační proces v roli serveru sám alokuje paměť pro datové struktury, které bude při komunikaci používat, a pak volá rutinu *t_listen*. Ta čeká na volání od klienta, a vrací řízení zpět v okamžiku, kdy takovéto volání přijde. Klient naopak využije k navázání spojení rutinu *t_connect*, s obdobným efektem jako v BSD Unixu.

.PI OBR63_3.TIF, 20, 40, 10

Obr. 63.3.: Představa práce s rozhraním TLI při spojované komunikaci

Pokud je server ochoten akceptovat přicházející volání (které představuje výzvu k navázání spojení), volá rutinu *t_accept*, která volání přijme, a zajistí navázání spojení s klientem. Pak již může následovat vlastní přenos dat (rutinami *t_rcv* a *t_snd*, nebo přímo operacemi *read* a *write*), a ukončení spojení (které může být realizováno buď jako "řádné", zaručují doručení všech již odeslaných dat, nebo jako ukončení "okamžité", které toto nezaručuje).

Případ nespojovaného způsobu komunikace mezi klientem a serverem ukazuje obrázek 63.4.: zde po alokaci paměti pro datové struktury ihned následuje vlastní přenos dat.

Zajímavou odlišností AT&T Unixu a BSD Unixu je to, zda server může odmítnou volání klienta, či nikoli. V BSD Unixu to možné není - systémové volání *accept* zde čeká na jakékoli volání, a to vždy přijme. Teprve pak se server dozvídá (z výstupních parametrů systémového volání *accept*), kdo mu vlastně zavolal, a pouze následně může již navázané spojení zase zrušit. Naproti tomu v AT&T Unixu má server možnost odmítnout určitého klienta, a spojení s ním vůbec nenavázat. Na volání klienta totiž čeká rutina *t_listen*, která s ním však nenaváže spojení, ale pouze vrátí potřebné informace o tom, kdo vlastně volá. Server se podle nich rozhodne buď volání přijmout (tj. navázat spojení, pomocí rutiny *t_accept*), nebo jej odmítnout (v tomto případě volá rutinu *t_snddis*).

.PI OBR63_4.TIF, 20, 40, 10

Obr. 63.4.: Představa práce s rozhraním TLI při nespojované komunikaci

64/ Aplikační vrstva TCP/IP

Ve třech minulých dílech jsme se zabývali rozhraním mezi transportní a aplikační vrstvou v rámci síťového modelu TCP/IP, neboli tím, jak mohou být služby transportní vrstvy zpřístupněny jednotlivým aplikacím. Ukázali jsme si, že zde velmi záleží na konkrétním operačním systému, a pak jsme se podrobněji věnovali dvěma možnostem, se kterými se můžeme setkat v prostředí operačního systému Unix. Dnes již dojde řada na vlastní aplikační vrstvou.

Když jsme si ve 38. dílu popisovali aplikační vrstvu v rámci referenčního modelu ISO/OSI, dospěli jsme k zajímavému závěru: aplikační vrstva tohoto modelu není určena k tomu, aby se v ní provozovaly jednotlivé aplikace. Na počátku, když se celková koncepce referenčního modelu teprve utvářela, se ještě předpokládalo, že jednotlivé aplikace budou provozovány přímo v aplikační vrstvě. Jakmile se ale začaly podrobněji rozpracovávat jednotlivé druhy aplikací a jejich konkrétní protokoly, zjistilo se, že na aplikační vrstvu zbývá ještě velmi mnoho úkolů. Kdyby si

je ovšem zajišťovala každá aplikace sama, bylo by to zbytečným plýtváním, protože by stejné funkce byly implementovány vícekrát. Proto se v rámci referenčního modelu dospělo k závěru, že je výhodnější tyto funkce implementovat tak, aby mohly být společné pro více aplikací. Z tohoto důvodu pak byly zbývající části vlastních aplikací (především jejich uživatelská rozhraní) "vysunuty" nad aplikační vrstvu, ve které naopak zůstaly jen jejich "podpůrné" prostředky (viz 38. díl).

Referenční model ISO/OSI tedy staví na předpokladu, že jednotlivé aplikace budou mít mnoho společného, a že se tedy vyplatí realizovat jejich společné části samostatně, a implementovat je jen jednou. Souvisí to ostatně i se způsobem, jakým aplikační protokoly v rámci referenčního modelu vznikaly - přístupem, který by bylo možné označit jako maximalistický. Často se totiž do aplikačních protokolů zahrnulo "všechno, co by mohlo být někdy někomu užitečné". Snad nejmarkantněji se tento přístup projevil na koncepci protokolu pro přenos souborů v rámci ISO/OSI (modelu FTAM - File Access, Transfer and Management), který je tak obsáhlý a komplikovaný, že snad nikdy nebude moci být implementován v celém svém rozsahu.

Naproti tomu síťový model TCP/IP vznikal méně "od zeleného stolu", než referenční model ISO/OSI, a více vycházel z praktických zkušeností a potřeb. Jeho aplikace začínaly jako relativně jednoduché, a teprve postupem času se jejich funkce a schopnosti zvětšovaly, a začaly se zavádět nové, náročnější druhy aplikací. Síťový model TCP/IP proto vychází spíše z předpokladu, že jednotlivé aplikace nebudou mít tolik společného, aby se tyto jejich společné části vyplatilo osamostatnit. Na rozdíl od referenčního modelu ISO/OSI proto očekává, že každá aplikace si sama zajistí to, co potřebuje a co ji nižší vrstvy neposkytují. Teprve v poslední době se pak i v rámci síťového modelu TCP/IP začínají některé podpůrné mechanismy v rámci aplikační vrstvy osamostatňovat (např. mechanismus volání vzdálených procedur - viz dále).

Zde je dobré si uvědomit další rozdíl mezi referenčním modelem ISO/OSI a síťovým modelem TCP/IP, který spočívá v počtu jejich vrstev. Referenční model ISO/OSI totiž zařazuje mezi transportní vrstvu a vrstvu aplikační ještě dvě další vrstvy (relační a prezentační), které také poskytují podpůrné služby vlastním aplikacím (kromě samotné aplikační vrstvy). Síťový model TCP/IP však nemá žádnou analogii relační a prezentační vrstvy ISO/OSI. Také jejich funkce si proto v prostředí TCP/IP musí zajistit jednotlivé aplikace vlastními silami - viz obr. 64.1.

.PI OBR64_1.TIF, 20, 40, 10

Obr. 64.1.: Aplikace a aplikační vrstva v RM ISO/OSI a v síťovém modelu TCP/IP

Stejně tak jako referenční model ISO/OSI, byl i síťový model TCP/IP navržen pro heterogenní prostředí, tedy pro prostředí počítačových sítí, jejichž uzlové počítače se mohou i dosti výrazně lišit - nejen použitým hardwarem, ale také například konvencemi pro znázorňování čísel, kódováním jednotlivých znaků, konvencemi operačních systémů o vlastnictví a přístupových právech k souborům apod. V referenčním modelu se o odstranění některých odlišností (hlavně ve vnitřních formátech) stará prezentační vrstva. V modelu TCP/IP je vše na samotných aplikacích.

.cp20

Klient vs. server

Většina aplikací a aplikačních protokolů v rámci síťového modelu TCP/IP vychází z modelu klient/server. Předpokládá tedy existenci dvou složek, které nemají rovnocenné postavení - klient žádá o konkrétní služby a je iniciátorem veškeré komunikace, zatímco server své služby neposkytuje z vlastní iniciativy, ale pouze na žádost klienta.

Rozeberme si tuto představu poněkud podrobněji. Představme si (viz též obr. 64.2.) složku v roli klienta na počítači A. Tato klientská složka formuluje své požadavky na konkrétní služby, a zasílá je své partnerské složce v roli serveru na počítači B. Složka v roli serveru je přijímá, zpracovává a reaguje na ně (např. vrací zpět požadovaná data). Přesný způsob komunikace přitom závisí na konkrétní aplikaci - některé využívají pro vzájemnou komunikaci klienta se serverem spolehlivé a spojované transportní služby protokolu TCP, zatímco jiné dávají přednost nespolehlivým a nespojovaným transportním službám protokolu UDP. Vždy však musí platit to, co jsme si již naznačovali v 55. dílu: že klient musí být schopen "najít" příslušný server. Složka v roli serveru proto musí přijímat požadavky svých klientů na takovém portu (přechodovém bodu mezi transportní a aplikační vrstvou), který je klientovi znám (tedy na tzv. dobře známém portu, viz 55. díl),

.PI OBR64_2.TIF, 20, 40, 10

Obr. 64.2.: Představa složek v roli klienta a serveru

.cp20

Implementace klienta a serveru v prostředí Unixu

Pro lepší představu o celkové filosofii aplikačních protokolů síťového modelu TCP/IP je vhodné si alespoň naznačit jejich implementaci v prostředí operačního systému Unix.

Ve víceúlohovém prostředí Unixu je vcelku přirozené koncipovat obě složky jako samostatné procesy - každý ovšem s jiným postavením v rámci operačního systému. Proces, realizující složku v roli serveru, je obvykle systémovým procesem. Tedy takovým procesem, který není spouštěn na popud uživatele, ale na popud operačního systému (obvykle již při jeho počátečním zavádění). Uživatel o jeho existenci vlastně ani nemusí vědět, protože takovýto proces s uživatelem přímo nekomunikuje. V prostředí Unixu se procesy tohoto typu označují jako **démoni** (**daemons**).

Naproti tomu proces, realizující klientskou složku, je často běžným typem aplikačního procesu, který si uživatel sám explicitně spouští až na základě skutečné potřeby. Kromě toho, že implementuje příslušné aplikační protokoly, bývá tento aplikační proces vybaven i vlastním uživatelským rozhraním - nejčastěji jednoduchým a řádkově orientovaným. Typickým příkladem může být protokol FTP (File Transfer Protocol), jehož klientská složka bývá v Unixu implementována jako samostatná utilita, s vlastním řádkově orientovaným uživatelským rozhraním. Prostřednictvím tohoto jednoduchého rozhraní pak uživatel mj. zadává, které soubory si přeje přenést.

V poslední době se však toto schéma začíná poněkud měnit, v souvislosti s tím, jak i Unix postupně získává grafická uživatelská rozhraní, a opouští uživateli tolik kritizovaná, strohá řádková rozhraní. Zde již jsou i klientské složky realizovány bez vlastního uživatelského rozhraní, a toto jim vytváří až příslušná grafická nadstavba (např. systém X-Window). S klientskou složkou, implementující vlastní aplikační protokoly, pracuje prostřednictvím přesně definovaných konvencí, které tvoří tzv. **aplikační programové rozhraní** (**API, Application programming Interface**).

Uvedené koncepci se však poněkud vymykají některé speciální aplikace v rámci síťového modelu TCP/IP. Zřejmě nejmarkantnějším příkladem je aplikační protokol NFS (Network File System, vyvinutý firmou Sun MicroSystems), případně jeho alternativní protokol RFS (Remote File Sharing, vyvinutý firmou AT&T). Ten má totiž za úkol zajišťovat plně transparentní sdílení souborů mezi uzlovými počítači, o kterém by uživatel vlastně ani neměl vědět. Na rozdíl od protokolu FTP, který je určen pro interaktivní přenos celých souborů, NFS své uživatelské rozhraní ani dost dobře mít nemůže. Místo toho musí být implementován tak (viz obr. 64.3.), aby mohl být operačním systémem "překryt", a dostával od něj k vyřízení ty požadavky na přístup k souborům, které nejsou lokální - tedy které se fyzicky nenachází na lokálním disku.

Protokol NFS je navíc relativně mladým protokolem v rámci TCP/IP, a při jehož vzniku se již uplatnila obdobná tendence, jako u referenčním modelu ISO/OSI: nenechat aplikace, aby si vše potřebné zajišťovaly samy, ale poskytnout jim potřebné podpůrné prostředky, realizované jako samostatné (na úrovni aplikační vrstvy), a tudíž využitelné i jinými aplikacemi. Protokol NFS proto počítá s tím, že bude využívat dva samostatně implementované mechanismy: mechanismus vzdáleného volání procedur (**RPC, Remote Procedure Call**), který mu zprostředkovává vhodný způsob komunikace s jeho partnerskou složkou v roli klienta, a dále mechanismus, zajišťující nezbytnou konverzi přenášených dat (**XDR, eXternal Data Representation**).

.PI OBR64_3.TIF, 20, 40, 10

Obr. 64.3.: Představa implementace klientské složky protokolu NFS

65/ Terminálové relace a vzdálené přihlašování

V minulém dílu jsme se začali zabývat celkovou koncepcí aplikačních služeb v sítích na bázi TCP/IP a naznačili si principy jejich začlenění do operačního systému (na příkladu Unixu). Dnes se již můžeme věnovat první konkrétní aplikaci, která umožňuje tzv. vzdálené přihlašování. Snad ale neuškodí si nejprve podrobněji naznačit, co tato možnost vlastně znamená a k čemu ji lze využít, a teprve pak se zabývat tím, jak je v prostředí TCP/IP sítí implementována.

Možnost **vzdáleného přihlašovaní** (v angličtině: **remote login**) úzce souvisí s existencí tzv. **terminálových relací** (anglicky: **terminal sessions**). Pro uživatele víceúlohových operačních systémů je to pojem jistě dobře známý, ale pro mnohé uživatele, "odkojené" počítači PC a jednoúlohovým prostředím MS DOSu, může představovat něco zcela nového a nepříliš "průhledného".

Od dávkového zpracování k interaktivnímu

Historicky nejstarším způsobem využití výpočetní techniky je dávkové zpracování (batch processing), při kterém uživatel sestavil svou úlohu, předem pro ni připravil veškerá její vstupní data a vše "zabalil" do jednoho celku - tzv. dávky. Ta pak byla ve vhodné době provedena. Obvykle byly jednotlivé dávky řazeny do front a každá dávka se provedla tehdy, až na ni došla řada. Uživatel tedy nebyl v přímém kontaktu se svou úlohou, nemohl bezprostředně reagovat na různé situace, ke kterým při provádění jeho úlohy mohlo docházet, a nemohl ani jinak interaktivně komunikovat se svou úlohou. V době, kdy se jeho úloha skutečně zpracovávala, nemusel vůbec být ve fyzické blízkosti příslušného počítače.

Obrat nastal až v okamžiku, kdy se přešlo na práci v režimu sdílení času. Nyní se již na jednom počítači mohlo současně nacházet více úloh v rozpracovaném stavu, a procesor velmi rychle přecházel z provádění jedné úlohy na provádění druhé úlohy, pak třetí atd. Díky tomuto rychlému střídání vznikla dostatečně věrná iluze toho, že procesor se věnuje více úlohám současně, tzn. že na jednom počítači běží najednou více úloh. Ve skutečnosti se avšak všechny tyto úlohy dělily o čas, který věnoval procesor jejich provádění. Jinými slovy: jednotlivé úlohy sdílely čas procesoru, odsud: režim **sdílení času** (**time sharing**).

Tímto způsobem (tj. v režimu sdílení času) bylo nadále možné provozovat úlohy připravené předem ve formě dávek, které nevyžadují bezprostřední styk se svým zadavatelem. Režim sdílení času však umožnil to, aby uživatelé byli v bezprostředním styku se svými úlohami a mohli s nimi komunikovat interaktivním způsobem.

Ukažme si na konkrétním příkladu, v čem byl pro uživatele rozdíl. Vžijme se do role programátora, který opravuje zdrojový text svého programu a má jej na daném počítači uložen jako jeden ze svých souborů. V případě dávkového zpracování musel tento uživatel předem připravit příkazy pro opravu všech chyb ve zdrojovém programu stylem: "na řádce XY změň řetězec ABCD na EFGH", přidat ještě příkaz pro nový překlad zdrojového textu, sestavit z těchto příkazů dávku, a tu zadat k provedení. Druhý den si přišel pro výsledek, zjistil nové chyby a celý postup se opakoval.

Na počítači pracujícím v režimu sdílení času může náš programátor pracovat odlišným způsobem. Nejprve začne komunikovat s operačním systémem, jemuž zadá příkaz ke spuštění interaktivního editoru. Jeho prostřednictvím pak postupně provede potřebné opravy v souboru se zdrojovým textem, načež práci s editorem ukončí a začne opět komunikovat přímo s operačním systémem. Tomu zadá příkaz k překladu opraveného zdrojového textu a nechá si výsledky tohoto překladu zobrazit. Odhalí-li překladač další chyby ve zdrojovém textu, uživatel si znovu zavolá editor a celý cyklus se opakuje. Tentokrát však již s "dobou obrátky", která se měří spíše na minuty než na celé dny, jako u dávkového zpracování.

Terminály a terminálové relace

Interaktivní způsob práce v systému se sdílením času má jeden důležitý předpoklad: operační systém i jednotlivé aplikační úlohy potřebují vhodný mechanismus komunikace se svým uživatelem. Zatímco v případě dávkového zpracování si úlohy své vstupní příkazy a data přebíraly ze samotné dávky a své vstupy posílaly tam, kam jim příkazy v dávce přikazovaly (obvykle do výstupních souborů, ev. na tiskárnu), v

případě interaktivní komunikace čekají, až jim uživatel zadá příslušné příkazy či jiné vstupy ze své klávesnice, a své výstupy mu posílají na displej, kde si je uživatel může přečíst.

To ale znamená, že operační systém i aplikační úloha musí mít přístup ke klávesnici a k displeji, které příslušný uživatel používá. Navíc, jde-li o víceuživatelské prostředí, které umožňuje interaktivní způsob práce více uživatelům současně, musí takovýchto dvojic "klávesnice a displej" existovat více a každá konkrétní úloha musí vědět, na které z nich právě pracuje ten "její" uživatel.

Zmíněná dvojice "klávesnice a displej" se obvykle označuje jako **terminál** a často tvoří i jeden konstrukční celek (i když to není zdaleka podmínkou). Počítače s víceuživatelským operačním systémem, které pracují v režimu sdílení času, bývají vybaveny větším počtem těchto terminálů.

Existence více než jednoho terminálu však znamená, že operační systém i jednotlivé aplikační úlohy nemohou počítat s tím, že uživatel s nimi bude komunikovat vždy z jednoho a téhož terminálu. Naopak je žádoucí, aby z pohledu uživatelů byly všechny terminály ekvivalentní - aby uživatel mohl přijít ke kterémukoli z nich, který je právě volný, a pracovat z něj přesné stejným způsobem jako z kteréhokoli jiného terminálu.

Vztah mezi úlohou, která odněkud očekává své vstupy a někam chce posílat své výstupy, a konkrétním terminálem, ze kterého se uživatel rozhodne pracovat, je tedy vztahem, který není á priori a pevně dán, ale utváří se až v okamžiku, kdy uživatel "zasedne" k některému z terminálů a projeví své přání pracovat v operačním systému.

Toto své přání projevuje uživatel tzv. **přihlášením** (anglicky: **login,** nebo: **logon**), kdy na výzvu operačního systému zadá své uživatelské jméno a svou identitu prokáže potřebným heslem.

Přihlášením uživatele do systému z určitého terminálu vzniká výše naznačený vztah mezi konkrétním terminálem a aplikační úlohou, který je označován jako **terminálová relace**. Nejlépe si ji lze představit jako abstraktní rozhovor, který určuje, kdo s kým rozmlouvá. Přihlášením uživatele vzniká takováto terminálová relace mezi terminálem a operačním systémem jako takovým - přesněji mezi terminálem a tou systémovou úlohou, která zajišťuje funkci interpreteru příkazů operačního systému, vysílá na obrazovku terminálu nápovědný znak (tzv. prompt), přijímá od uživatele jeho příkazy pro operační systém, a zajišťuje jejich provedení (v prostředí Unixu se takováto úloha označuje jako **shell**).

Příkazem, jejž uživatel operačnímu systému zadává, může být například příkaz ke spuštění určité aplikace, která pak přebírá již existující terminálovou relaci a jejím prostřednictvím komunikuje se svým uživatelem. Jde-li například o editor, zobrazuje v rámci terminálové relace obsah editovaného souboru na displeji příslušného terminálu a z jeho klávesnice přijímá své příkazy.

V tuto chvíli je vhodné si naši postupně krystalizující představu terminálové relace ještě poněkud poopravit. Není totiž zcela přesné představovat si, že z jednoho terminálu je možné zřídit (někdy se říká též: otevřít) nejvýše jednu terminálovou relaci. V principu je možné, aby si uživatel z jednoho terminálu otevřel více relací s různými aplikacemi, či dokonce s aplikací jedinou - například aby si v každé jednotlivé relaci spustil tentýž editor (přesněji: instanci téhož editoru) a editoval jiný soubor. V každém okamžiku jsou sice klávesnice a displej terminálu vyhrazeny jen

jedné relaci - tj. uživatel vždy komunikuje jen s jednou úlohou v rámci jedné relace - ale pomocí klávesnice se může mezi jednotlivými relacemi rychle přepínat (obvykle pomocí nějaké horké klávesy). Záleží jen na tom, zda daný operační systém a příslušný terminál vychází této možnosti vstříc.

Další zajímavé možnosti se pak otevírají v okamžiku, kdy do hry vstupují vzájemně propojené počítače a počítačové sítě.

Vzdálené terminálové relace

Až doposud jsme mlčky předpokládali, že terminálová relace se otevírá mezi úlohami běžícími na určitém počítači a konkrétním terminálem, který je k tomuto počítači připojen. Ve skutečnosti však jde jen o jednu z možností, která se neformálně označuje jako "lokální" či "místní" terminálová relace. Jejím charakteristickým rysem je skutečnost, že v ní vystupuje terminál přímo připojený k danému počítači. Není přitom podstatné zda je toto připojení realizováno krátkým kabelem (terminál se nachází ve fyzické blízkosti počítače), nebo zda je pro připojení terminálu využit například pevný telefonní okruh (vlastní terminál se nachází na druhém konci města).

.PI OBR65_1.TIF, 20, 40, 10

Obr. 65.1: **Vzdálené vs. lokální terminálové relace**

Jiná situace nastává v případě počítačů vzájemně propojených prostřednictvím počítačové sítě. Zde je totiž možné realizovat tzv. **vzdálené terminálové relace** (**remote terminal session**), při kterých vzniká terminálová relace ve výše uvedeném smyslu mezi terminálem jednoho počítače, a úlohou běžící na jiném počítači. Představme si - v souladu s obrázkem 65.1 - uzlové počítače A a B a jejich terminály Ta a Tb. Uživatel počítače A, který pracuje u terminálu Ta, se nejprve přihlásí do systému na počítači A (tj. provede tzv. login). Tím vytvoří "lokální" terminálovou relaci mezi terminálem Ta a počítačem A a v rámci táto relace bude komunikovat s interpreterem příkazů operačního systému počítače A (na obrazovce se mu tedy objeví nápovědný znak - tzv. prompt - operačního systému počítače A). Uživatel však zadá příkaz ke spuštění zvláštní aplikační úlohy (říkejme jí pracovně "úloha telnet"), která zajistí následující věci: nejprve naváže spojení s počítačem B (obecně s tím počítačem, který jí uživatel zadá) a pak zřídí terminálovou relaci mezi terminálem Ta a počítačem B. Přesněji: mezi terminálem Ta a interpreterem příkazů operačního systému počítače B. V rámci této terminálové relace, která si již zaslouží přívlastek "vzdálená", se veškeré vstupy z klávesnice terminálu Ta dostávají nejprve k "úloze

telnet" na počítači A, ale ta je ihned odesílá na počítač B, kde příslušná data vystupují ve stejné roli jako vstupy z kteréhokoli "místního" terminálu. Naopak veškeré výstupy, které jsou v rámci této relace generovány, jsou odesílány na počítač A, kde je přijímá "úloha telnet" a zajišťuje jejich okamžité zobrazení na displeji terminálu Ta (viz obr. 65.2).

.PI OBR65_2.TIF, 20, 40, 10

Obr. 65.2: **Představa realizace vzdálené terminálové relace**

Terminál Ta fyzicky připojený k počítači A se tak začne chovat jako terminál přímo připojený k počítači B. Na displeji terminálu se nejprve zobrazí nápovědný znak (prompt) interpreteru příkazů počítače B a uživatel pak může z terminálu Ta zadávat příkazy operačnímu systému na počítači B. Nejprve se tedy přihlásí do systému na počítači B (provést tzv. **vzdálené přihlášení**, angl.: **remote login**) a pak si například spustí nějakou aplikační úlohu. Ta "poběží" na počítači B, ale své vstupy bude očekávat z terminálu Ta, a její výstupy se budou zobrazovat na displeji terminálu Ta. Uživatel, který sedí u terminálu Ta, tak bude mít stejné postavení a stejné možnosti práce na počítači B, jako kdyby seděl přímo u terminálu Tb tohoto počítače.

K ocenění výhodnosti a užitečnosti vzdálených terminálových relací a vzdáleného přihlašování snad zbývá dodat jen jediné: je vcelku lhostejné, zda se počítače A a B nalézají blízko sebe nebo zda jsou například každý v jiném městě či na jiném kontinentě. Jedinou podmínkou pro možnost vzdáleného přihlašování je vhodné propojení obou počítačů a pak také softwarová podpora vzdálených terminálových relací. V případě počítačových sítí je potřebné spojení úkolem nižších vrstev a podpora vzdálených terminálových relací je jednou z aplikačních služeb. O tom, jak jsou vzdálené terminálové relace konkrétně realizovány v sítích na bázi TCP/IP, si povíme příště.

66/ Virtuální terminály

Praktická implementace vzdáleného přihlašování a vzdálených terminálových relací, kterými jsme se zabývali minule, naráží na nemalé problémy. Podívejme se nejprve, v čem je podstata těchto problémů, a pak si ukažme, jaké jsou možnosti jejich řešení.

Dnešní počítačový svět není homogenní. Byl, je a zřejmě vždy bude obydlen výpočetními systémy, které vychází z odlišných filosofií, používají odlišné přístupy a metody, technologie a techniky, a v důsledku toho nejsou nikdy zcela kompatibilní. Snahy o co největší "otevřenost" však vedou alespoň k tomu, že různé systémy spolu dokáží rozumným způsobem spolupracovat. Tendence ke vzájemnému sbližování, zajišťovaná dodržováním společných standardů, se však neprosazuje ve všech oblastech stejně snadno.

Každý terminál je jiný

Oblastí, která je vůči všeobecnému trendu ke standardizaci snad nejvíce imunní, je právě výroba terminálů. Jednotliví výrobci se totiž doslova předhání v tom, jakými schopnostmi dokáží své výrobky vybavit. To by samo o sobě bylo jistě velmi vítané, ale háček je v tom, že neexistuje všeobecný konsensus o tom, jak to konkrétně dělat. Na obzoru není žádný všeobecně uznávaný a závazný standard (standard de jure), který by říkal, co všechno má takový terminál umět, jak přesně má fungovat a jak se má konkrétně ovládat. Místo toho si každý výrobce řeší věci po svém, a tak vznikají pouze standardy de facto, které svými výrobky vytvářejí nejrenomovanější výrobci, a ostatní výrobci se jim pak ve vlastním zájmu přizpůsobují. I tak se ale mohou dosti podstatně lišit i různé druhy terminálů od jednoho a téhož výrobce.

Zkusme si nyní naznačit, v čem mohou rozdíly mezi jednotlivými terminály spočívat.

Největší rozdíly mohou být již v samotné koncepci terminálu a "míře inteligence", kterou je vybaven. Z tohoto pohledu se terminály obvykle rozdělují do tří skupin, na tzv. **znakové** (někdy též: **řádkové terminály**, anglicky: **scroll mode terminals**), na **stránkové** (**page mode terminals**) a **formulářové** (**form mode terminals**).

Znakový terminál je nejjednodušším typem, a je charakteristický tím, že neumí pohybovat kurzorem nahoru a dolů po obrazovce. Jeho displej se chová obdobně jako tiskárna - každý znak je vytisknut (zobrazen) tam, kde se právě nachází kurzor, a ten je následně posunut doprava na následující pozici. Pokud by přitom přešel přes pravý okraj řádky, je přesunut na začátek řádky nové, a pokud by pro tuto novou řádku již nebylo na displeji místo, jsou všechny právě zobrazované řádky odrolovány (anglicky: scrolled) směrem nahoru. Znakový terminál tedy neumožňuje "vrátit" se zpět (posunout kurzor) na dříve zobrazenou řádku (ale obvykle umožňuje pohybovat kurzorem zpět v rámci dané řádky). Pro ovládání znakového terminálu jsou pak zapotřebí příkazy typu: zobraz znak (na pozici kurzoru, s následným posunutím kurzoru na další pozici), přejdi na novou řádku, případně: vrať kurzor o jednu pozici zpět (v rámci dané řádky), vymaž znaky od kurzoru do konce řádky apod.

Při určitém zjednodušení si lze představit, že znakový terminál má jen jednu jedinou řádku (zatímco na displeji je zobrazováno ještě několik "předchozích" řádek, které ale již nejsou jinak přístupné).

Naproti tomu u stránkového (page mode) terminálu si můžeme představit, že jeho displej reprezentuje dvojrozměrné pole o n řádcích a m sloupcích, a kurzor lze nastavit na libovolnou pozici v tomto dvojrozměrném poli.

Díky této možnost je pak možné pohybovat kurzorem všemi směry, a na rozdíl od znakového terminálu je již možné implementovat na stránkovém terminálu například celoobrazovkové (full-screen) editory a další aplikace, které pohyb kurzoru všemi směry vyžadují. Z pohledu ovládání pak přibývají u stránkového terminálu nezbytné příkazy pro nastavování pozice kurzoru.

Formulářový terminál je ještě "chytřejším" typem terminálu. U něj si můžeme představit, že místo jednoho dvojrozměrného pole se jeho displej dokáže chovat jako několik (menších) jedno a dvojrozměrných polí. Ty pak můžeme interpretovat jako navzájem disjunktní "políčka" formuláře, se kterými se pracuje nezávisle na sobě (tj. která se samostatně vyplňují). Formulářový terminál pak obvykle umožňuje definovat strukturu "formuláře": stanovit počet, velikost, dimenzi a druh jednotlivých polí, způsob jejich zobrazení na displeji (včetně doplnění vysvětlujícím textem) atd.

I v rámci jedné a téže kategorie (např. znakových terminálů) se však jednotlivé druhy terminálů mohou výrazně lišit, a to ve dvou základních směrech:

- v konkrétních parametrech a dílčích schopnostech. Rozdílný může být například počet znakových pozic na řádce, počet zobrazovaných řádek, schopnost používat alternativní sady znaků a různé efekty při jejich zobrazování (podtržený text, tučné písmo, písmo dvojnásobné výšky, šířky apod.)

- v konkrétním způsobu ovládání. Některé terminály například očekávají, že konec řádky bude vyznačen dvojicí znaků CR a LF, zatímco jiné znaky očekávají pouze znak CR, a ještě jiné pak pouze znak LF. Odlišné bývají i nejrůznější další řídící znaky, například pro nastavení kurzoru na zadanou pozici (u stránkových terminálů) či jeho posun na předchozí pozici, i samotný efekt jimi vyvolaných akcí (zda se např. při posunu kurzoru o jednu pozici zpět žádný znak nemaže, či naopak ano). Pokud terminál "umí" různé další efekty, jako například podtrhovat, psát tučně, používat jinou znakovou sadu apod., tyto jeho schopnosti se obvykle ovládají celými posloupnostmi řídících znaků (označovanými také jako tzv. **escape sekvence**). Snad u každého terminálu jsou ale jiné.

Terminálová emulace

Pro správné pochopení všech problémů kolem používání terminálů je vhodné si také naznačit něco o jejich vývoji. Historicky nejstarším typem terminálu bylo vlastně dálnopisné zařízení (anglicky: **teletype**), které místo dnešního displeje mělo tiskárnu. Každý znak, zadaný z klávesnice, byl okamžitě odeslán druhé straně, a každý znak, který od druhé strany přišel, byl ihned vytisknut. Dalším vývojovým stádiem již byly znakové terminály, namísto tiskárny vybavené displejem. Stále však to byla jednoúčelová zařízení, schopná fungovat právě a pouze jako terminál. Postupným vývojem získávaly tyto terminály nové a nové schopnosti, ze znakových se staly stránkové a posléze i formulářové terminály, ale svou jednoúčelovou povahu si držely stále.

V poslední době, zvláště pak s masovým nástupem počítačových sítí, se však začíná používat i jiné řešení: **terminálová emulace** (**terminal emulation**). Vše je postaveno na myšlence, že místo jednoúčelového zařízení s pevně danou funkcí se použije zařízení univerzální, které se pouze bude chovat stejně jako jednoúčelový terminál (bude jej tzv. emulovat, odsud: emulace). Vhodným kandidátem na tuto funkci je osobní počítač, který není obtížné naprogramovat tak, aby věrně napodoboval chování v podstatě jakéhokoli terminálu.

Výhodou terminálové emulace není jen její flexibilita - tedy schopnost přizpůsobit se podle potřeby vlastnostem více různých terminálů. Další výhodou je i to, příslušné univerzální zařízení (osobní počítač) může vystupovat v roli emulovaného terminálu jen v době, kdy si to jeho uživatel přeje, zatímco v ostatní době může sloužit jiným účelům. Aplikační programy, zajišťující emulaci terminálů, také obvykle umožňují otevírat více terminálových relací současně - jak jsme si ostatně již naznačovali v minulém dílu.

Zajímavé jsou pak i ekonomické ukazatele. Díky dnešní masové výrobě osobních počítačů se jejich cena stala srovnatelnou s cenou jednoúčelových terminálů.

Jak se s různorodostí vyrovnat

Již v minulém dílu jsme si naznačili jednu velmi důležitou skutečnost: ve víceúlohovém prostředí na počítači, vybaveném více terminály, nemohou jednotlivé úlohy předem znát, se kterým terminálem budou v rámci terminálových relací skutečně pracovat. Konkrétní typ všech terminálů daného počítače sice může být předem znám, ale připustíme-li možnost vzdálených terminálových relací, uskutečňovaných prostřednictvím počítačové sítě, není již možné předpokládat prakticky nic.

Jednotlivé aplikace, které v rámci terminálových relací pracují s terminály, je však nutně musí umět ovládat. Musí vědět, co všechno terminál "umí", jaké znaky je schopen přijímat a zobrazovat, s jakými řídícími znaky pracuje, jak mají být zakončovány jednotlivé řádky, jaké používá řídící sekvence apod.

Každá aplikace si samozřejmě může předepsat, že chce pracovat jen s takovým a takovým typem terminálu, a tomu se plně přizpůsobit. To je ale příliš omezující vůči uživatelům, kteří disponují jinými terminály, zvláště pak v prostředí sítí. Stejně tak není dost dobře možné, aby se jednotlivé úlohy přizpůsobovaly určitému konkrétnímu repertoáru terminálů - vždy se totiž najde někdo, kdo bude chtít používat ještě jiný terminál, a navíc: nové druhy terminálů, s novými schopnostmi, dokonalejšími vlastnostmi a hlavně odlišnými způsoby ovládání neustále vznikají.

Podstata virtuálních terminálů

Vzájemné přizpůsobení všech aplikací všem možným terminálům tedy není dost dobře možné (a nebylo by jistě ani příliš rozumné). Jaká je ale jiná možnost?

V podstatě jedinou schůdnou alternativou je vytvořit jednotný mezistupeň, a přizpůsobení typu "každý s každým" nahradit mnohem schůdnější variantou, při které se všechny aplikace přizpůsobují jednotnému mezistupni, kterému se na druhé straně přizpůsobují i všechny konkrétní terminály. Jak ale správně interpretovat zmíněný mezistupeň?

Z pohledu aplikací jde vlastně o terminál, který se vždy "tváří" stejně - tedy který má vždy stejné schopnosti, stejné vlastnosti, a stejný způsob ovládání. Ve skutečnosti však nejde o fyzické zařízení, ale například jen o určité konvence (formát dat a příkazů), či o objekt programové povahy (např. nějakou datovou strukturu), který skutečný terminál pouze zastupuje. Proto se také společnému mezistupniřiká **virtuální terminál** (**virtual terminal**).

V prostředí počítačových sítí a vzdálených terminálových relací pak není problémem, aby se společnému mezistupni (virtuálnímu terminálu) přizpůsobily i jednotlivé konkrétní terminály. Vzpomeňme si na minulý díl, kdy jsme si ukazovali, jak všechny vstupy a výstupy z konkrétního terminálu na daném počítači zpracovává "úloha telnet", a dále je posílá po síti do vzdáleného počítače. Není jistě těžké vytvořit tuto "úlohu telnet" tak, aby zajišťovala potřebné přizpůsobení konkrétního terminálu terminálu virtuálnímu.

Parametrický model virtuálního terminálu

Hlavním problémem právě naznačeného mechanismu je nalezení takové konkrétní formy (modelu) virtuálního terminálu, která by dokázala "pokrýt" co nejširší spektrum různých terminálů, ale současně umožňovala co nejlépe využít jejich nejrůznějších schopností. Tedy takového modelu, který by neomezoval "individualitu" jednotlivých terminálů více, než je opravdu nezbytně nutné.

Nejjednodušším modelem virtuálního terminálu je tzv. **parametrický model**. Zde si můžeme představit, že virtuální terminál je reprezentován tabulkou, která má pevně daný počet položek i pevně daný význam těchto položek. Tyto položky přitom mohou obsahovat například údaj o tom, kolik znakových řádek má displej terminálu, kolik znakových pozic je na jedné řádce, zda terminál očekává zakončení každé řádky dvojicí CR a LF apod. Dále zde mohou být příkazy (escape sekvence) pro ovládání nejrůznější efektů a možností (např. podtrhování, tučné písmo apod.), nastavování kurzoru (u stránkových terminálů) atd.

Vlastní aplikace se pak přizpůsobuje společnému mezistupni (virtuálnímu terminálu) tím, že skutečný terminál ovládá pouze "prostřednictvím" této tabulky. Jestliže například potřebuje nastavit kurzor na určitou pozici, najde si na příslušném místě v tabulce konkrétní řídící příkaz, a ten pak zadá terminálu. Konkrétní terminál se zase přizpůsobuje virtuálnímu terminálu tak, že jeho předem definované položky naplní konkrétnímu údaji, které přesně definují jeho vlastnosti a způsob ovládání.

Právě naznačený mechanismus se ve své nejobecnější podobě prosadil v BSD Unixu. Zde je výše naznačených tabulek více - po jedné pro každý druh terminálu, který daný operační systém zná - a jsou všechny sdruženy do jedné databáze, označované jako **termcap** (což je zkratka od: **terminal capabilities**). Aplikace, která chce korektním způsobem pracovat s terminálem, si nejprve musí zjistit, o jaký konkrétní druh terminálu se jedná. Pak si z databáze termcap "přečte" odpovídající tabulku, a příslušný terminál ovládá prostřednictvím příkazů (escape sekvencí), které v této tabulce najde. Správce systému přitom může databázi termcap doplňovat, a tím rozšiřovat repertoár terminálů, se kterými budou aplikace schopny plnohodnotně pracovat.

Za zvláštní případ parametrického modelu můžeme považovat takovou situaci, kdy bude pevně dána nejen struktura položek tabulky a jejich význam, ale bude pevně dán také jejich obsah. Tato představa odpovídá tomu, že konkrétní způsob ovládání virtuálního terminálu je fixován, po síti jsou přenášeny vždy stejné řídící příkazy, a potřebné přizpůsobení je zajišťováno až v místě, kde je připojen skutečný terminál ("úlohou telnet" z minulého dílu). Právě tato varianta je použita v síťovém modelu TCP/IP.

Objektový model virtuálního terminál

Parametrický model je použitelný pro znakové, a do značné míry i pro stránkové terminály. Pro formulářové terminály je ovšem zapotřebí poněkud obecnější model, kterým je model **objektový**. Ten si můžeme představit opět jako určitou tabulku, definující schopnosti konkrétního terminálu - v případě objektového modelu je ale tato tabulka označována jako **profil**. Kromě profilu ovšem tvoří virtuální terminál i obecnější datová struktura (objekt), která funguje jako vyrovnávací paměť mezi aplikací a terminálem. Aplikace, která chce něco zobrazit na displeji terminálu, to jednoduše zapíše do zmíněné datové struktury, a o další se již nestará. Je pak na skutečném terminálu (resp. jeho ovladači), aby obsah této datové struktury průběžně zobrazoval.

Zmíněná datová struktura přitom zdaleka nemusí být jen dvojrozměrné pole, reprezentující znakové pozice skutečného displeje. Například u formulářového terminálu to mohou být jednotlivá dílčí pole, představující části formuláře, a jiná část

datové struktury zase může definovat, jak mají být jednotlivá pole "poskládána" na skutečném displeji.

Tímto mnohem obecnějším způsobem jsou pak řešeny virtuální terminály v rámci referenčního modelu ISO/OSI.

67/ Telnet - I.

V předchozích dvou dílech jsme si vysvětlili podstatu terminálových relací a vzdáleného přihlašování v počítačových sítích, a posléze narazili na nejvážnější problém kolem jejich praktické implementace - na velkou různorodost používaných terminálů. Dnes se již můžeme začít podrobněji zabývat protokolem TELNET, který má v síťovém modelu TCP/IP realizaci vzdálených terminálových relací na starosti.

Každý aplikační protokol, který chce realizovat vzdálené terminálové relace v prostředí počítačových sítí, musí předpokládat určitou míru distribuovanosti své vlastní implementace - jak jsme si ostatně již naznačovali v 65. dílu, kde jsme se nad realizací vzdálených terminálových relací zamýšleli poprvé. Snad neuškodí si tuto představu ještě jednou oživit, tentokráte již na konkrétním příkladu protokolu TELNETu.

TELNET klient a server

Protokol TELNET předpokládá vzájemnou spolupráci dvou svých složek, jejichž role a postavení vychází z architektury klient-server: jedna složka, provozovaná na "lokálním" počítači (tj. na tom, ke kterému je fyzicky připojen uživatelův terminál) je označována jako TELNET klient, zatímco druhá složka, na "vzdáleném" počítači (ke kterému se prostřednictvím vzdálené terminálové relace uživatel přihlašuje) je v postavení TELNET serveru. Úloha obou složek přitom odpovídá představě, kterou jsme si zavedli v 65. dílu: složka v roli klienta přijímá všechny vstupy od terminálu, a odesílá je po síti své partnerské složce v roli serveru na "vzdáleném" počítači. Tato složka pak příslušné vstupy "podstrkuje" svému okolí tak, jako kdyby šlo o vstupy terminálu, připojeného k místnímu vzdálenému počítači. Filosofie protokolu TELNET přitom předpokládá, že obě složky mají formu aplikačních programů, a nejsou **tedy** "pevně zabudovány" v operačním systému - jak by asi bylo, alespoň u složky v roli serveru, zřejmě nejpřirozenější. Výhodou je totiž mnohem větší flexibilita a snadnost provádění případných změn, nevýhodou pak menší efektivnost. Uvědomme si totiž, že při takovémto řešení každý jednotlivý znak prochází celkem pětkrát "skrz" celý operační systém: poprvé než se dostane od terminálu ke klientské složce, podruhé

když jej tato složka odesílá na vzdálený počítač, potřetí když projde operačním systémem na tomto vzdáleném počítači až ke složce v roli serveru, počtvrté prochází operačním systémem, když jej serverová složka "podstrkuje" zpět svému operačnímu systému, a konečně popáté, když jej tento operační systém předává jiné aplikační úloze, která je konečným příjemcem vstupu. Analogicky pro všechny výstupy, které cestují opačným směrem.

Dále je pro toto řešení nutné, aby operační systém (alespoň na straně serveru) vycházel vstříc výše citované možnosti "podstrkování" vstupů, a poskytoval za tímto účelem vhodný mechanismus. Zde velmi záleží na konkrétním prostředí, ve kterém je protokol TELNET implementován. Operační systém Unix s takovouto možností počítá, a nabízí k využití zvláštní vstupně/výstupní body, na které se nejrůznější programy mohou napojit, a napodobovat chování skutečného terminálu. V AT&T Unixu se těmto vstupním bodům do operačního systému říká **pseudoterminály**, zatímco v BSD Unixu jsou označovány jako **pseudo tty**.

Za zmínku také stojí konkrétní způsob implementace obou složek v prostředí Unixu. Složka v roli klienta je běžným aplikačním programem, který si uživatel sám a explicitně spouští až v okamžiku, kdy to skutečně potřebuje. Naproti tomu složka v roli serveru má postavení systémového procesu - tzv. **TELNET démona** (viz 64. díl), označovaného též: **TELNETD.**

Nejen Unixem živ je TELNET

Protokol TELNET je relativně velmi jednoduchým protokolem, který se záměrně snaží nevázat na vlastnosti, schopnosti a služby určitého konkrétního prostředí. Ukažme si smysl tohoto počínání na příkladu: když se uživatel přihlásí k práci v operačním systému určitého počítače (tj. provede tzv. login), a poté si prostřednictvím TELNET-u otevře vzdálenou terminálovou relaci s jiným počítačem, musí se na něm znovu sám přihlásit (opět provést tzv. login). Bylo by sice možné, aby za něj toto přihlášení provedl TELNET automaticky. Znamenalo by to ovšem, že by musel vědět jak - musel by znát konkrétní konvence pro přihlašování a zadávání hesel, způsob uchovávání informací o uživatelích a jejich heslech a mnoho dalších konkrétních informací, které se v různých systémech mohou výrazně lišit.

Díky tomu, že se TELNET o automatické přihlašování nesnaží, může být implementován v prostředí různých operačních systémů, a to dokonce i takových, které pojem terminálových relací a uživatelských jmen a účtů vůbec neznají. Například je možné (a v současné době i velmi časté), aby klientská složka protokolu TELNET byla provozována jako běžný aplikační program na osobním počítači v prostředí operačního systému MS DOS, zatímco složka v roli serveru běžela na Unixovském počítači. Uživatel osobního počítače si pak může zřizovat vzdálené terminálové relace z prostředí DOSu do prostředí Unixu.

S implementací klientské složky protokolu TELNET se dnes můžeme setkat v prostředí snad každého operačního systému, zdaleka ne jen MS DOSu, a ze všech těchto operačních systémů je pak možné si zřizovat vzdálené terminálové relace s Unixovskými počítači. Zajímavé je, zda to platí i obráceně - tedy zda i složka v roli

serveru může být implementována v jiném prostředí než v Unixu, a zda je tedy možné si prostřednictvím protokolu TELNET zřizovat vzdálené terminálové relace i s jinými, než Unixovskými počítači. Odpověď je samozřejmě kladná, jen příslušných implementací TELNET serverů je zatím poněkud méně.

TELNET server na PC

Existují dokonce i implementace serverových složek protokolu TELNET pro operační systém MS DOS počítačů PC (dokonce z kategorie public domain). To je poněkud pikantní, protože MS DOS je jednoúlohový a jednouživatelský operační systém, který pojem terminálové relace vůbec nezná. Zřízení vzdálené terminálové relace s počítačem PC (pod MS DOSem) je pak vlastně formou dálkového ovládání (remote control) počítače PC jako celku. Pro potřeby dálkového ovládání počítačů PC dnes existuje celá řada komerčních i public domain programů, které ale vesměs vyžadují vlastní, specifické klientské složky. Výhodou tohotořešení (tj. TELNET serveru na počítači PC) je možnost využít standardní klientskou složku protokolu TELNET, která může být navíc provozována i jinde, než jen v prostředí MS DOSu. Díky tomu je možné, aby například Unixovský počítač na dálku ovládal počítač PC. Navíc je vhodné si uvědomit, že po zřízení vzdálené terminálové relace prostřednictvím protokolu TELNET je její existence pro uživatele transparentní, takže například uživatel Unixovského počítače pak může mít dojem, že pracuje na počítači PC. Nebude to ale iluze dokonalá - již jen z důvodu omezené přenosové rychlosti. Další problémy pak způsobuje např. odlišnost v počtu řádků: zatímco terminály Unixovských počítačů mají nejčastěji 24 řádků, počítače PC zobrazují v základním textovém režimu 25 řádek.

Rlogin je jiný

TELNET není zdaleka jediným protokolem pro realizaci vzdálených terminálových relací - v prostředí TCP/IP sítí je ale zřejmě nejrozšířenější. Existuje řada dalších, mnohem propracovanějších a komplexnějších protokolů pro vzdálené terminálové relace, které nabízí větší rozsah služeb, ale za svou dokonalost platí omezenějšími možnostmi nasazení, než jednodušší TELNE. Za zmínku stojí alespoň jeden alternativní protokol - **rlogin**. Pochází z prostředí BSD Unixu, a liší se od TELNETu především v tom, že "vnímá" prostředí, ve kterém pracuje jeho klientská i serverová složka, a snaží se využívat specifické vlastnosti a schopnosti těchto prostředí - např. pro zajištění automatického přihlašování. Přístup protokolu rlogin je takový, že správce systému na určitém počítači má možnost specifikovat, které jiné počítače považuje za "důvěryhodné" - v tom smyslu, že když uživatel již jednou prošel na takovémto počítači procesem ověřování své identity (zadáním uživatelského jména a hesla), daný počítač již nepožaduje nové opakování tohoto procesu. Jinými slovy: "věří" v identitu uživatelů, kteří mu vysílají své žádosti o zřízení vzdálených terminálových relací z těch počítačů, které jsou správcem určeny jako "důvěryhodné" (anglicky: trusted hosts), a nesnaží se ji ověřovat znovu. Dále má správce systému možnost stanovit, že konkrétní uživatelé "důvěryhodných" počítačů mají na daném počítači taková a taková uživatelská jména. Když pak přijde žádost o zřízení terminálové relace od konkrétního uživatele "důvěryhodného" počítače, může být tento uživatel automaticky přihlášen do systému daného počítače pod příslušným jménem, a bez nutnosti zadávat heslo.

Jakmile je toto možné, lze implementovat i takovou službu, která uživateli jednoho počítače umožní zadat příkaz, který se má provést na jiném počítači. Přitom se automaticky zřídí vzdálená terminálová relace, v rámci které se vzdálenému počítači zadá příslušný příkaz, a po jeho provedení se terminálová relace zase zruší. Uživateli přitom stačí zadat jen dva základní údaje: příkaz, který má být proveden, a dále jméno počítače, na kterém se tak má stát.

68/ Telnet - II.

V minulém dílu jsme se zabývali celkovou architekturou protokolu TELNET a jeho vazbou na prostředí, ve kterém je implementován. Ukázali jsme si, že tato vazba je zcela záměrně minimální, a díky tomu není protokol TELNET vázán na určitý operační systém. Dnes se již budeme věnovat konkrétním vlastnostem protokolu TELNET a mechanismům, které používá.

Nejprve si ale stručně připomeňme pojem virtuálního terminálu, kterému jsme věnovali 66. díl seriálu: vzhledem k velké různorodosti skutečných terminálů není dost dobře možné, aby se každá aplikace přizpůsobovala každému jednotlivému terminálu. Místo toho se zavedl tzv. virtuální terminál jako jednotný mezistupeň, kterému se z jedné strany přizpůsobují všechny aplikace, a z druhé strany všechny skutečné terminály. Jako "virtuální" je tento společný mezistupeň označován proto, že je ve skutečnosti jen abstrakcí, a nikoli fyzicky existujícím zařízením. V 66. dílu jsme si také naznačili, že protokol TELNET používá zvláštní případ tzv. parametrického modelu virtuálního terminálu.

NVT - Network Virtual Terminal

Virtuální terminál, používaný protokolem TELNET, je označován jako **NVT**, neboli **Network Virtual Terminal** (doslova: síťový virtuální terminál). Je zvláštním případem parametrického modelu v tom smyslu, že předpokládá pevně danou hodnotu jednotlivých parametrů, které terminál definují (viz 66. díl), čímž vlastně fixuje jeho vlastnosti i konkrétní způsob ovládání. Popišme si nyní, jaká je představa tohoto zařízení:

NVT je obousměrné, znakově orientované zařízení, které lze nejlépe přirovnat ke dvojici klávesnice-tiskárna. Klávesnice generuje jednotlivé znaky v kódu ASCII, zatímco tiskárna je průběžně tiskne. Celek pak odpovídá představě tzv. **znakového**, resp. **řádkového terminálu** (**scroll mode terminal**, viz 66. díl).

Ačkoli protokol TELNET využívá pro přenos dat spolehlivé a spojované přenosové služby transportního protokolu TCP, které vytváří plně duplexní spojení, NVT toto spojení využívá jen v poloduplexním režimu. Díky tomu je pak možné vystačit i s takovým skutečným terminálem, který je fyzicky poloduplexní (jako např. terminál IBM 2741).

NVT dále předpokládá, že přenos dat bude tzv. bufferován - tedy že data nebudou vysílána po jednotlivých znacích, ale že se budou nejprve hromadit ve vhodných vyrovnávacích pamětech (anglicky: buffers), a skutečně vysílány pak budou až větší celky. Jaké celky to ale mají být?

U řádkového terminálu, jakým je NVT, je jednoznačným kandidátem řádka. Délka řádky ovšem není pevně stanovena, a to ani u tiskárny. Na straně klienta určuje skutečnou délku každé jednotlivé řádky uživatel, který pracuje na příslušném terminálu - tím, že v určitý okamžik zmáčkne tlačítko ENTER, RETURN, NEW LINE či jak se na jeho terminálu jmenuje klávesa, kterou se zadává konec řádky. Klientská složka protokolu TELNET, která zpracovává uživatelův vstup z klávesnice, může na základě zmáčknutí této klávesy obdržet různý kód (např. jen znak CR, jen znak LF, dvojici znaků CR a LF apod.), podle konkrétního použitého terminálu. Sama však musí na jeho základě vygenerovat dvojici znaků CR a LF, neboť virtuální terminál NVT počítá s tím, že řádky budou zakončovány právě tímto způsobem. Analogicky je tomu i na straně serveru - aplikační proces, který generuje data, určená k zobrazení na terminálu, je člení na jednotlivé řádky takovým způsobem, jaký předpokládá příslušná "místní" konvence. Serverová složka protokolu TELNET je však před odesláním překládá do takového tvaru, aby byly zakončeny dvojicí CR LF.

Protokol TELNET tedy předpokládá, že data budou standardně přenášena po celých řádcích. To ale nemusí být vždy možné - není-li délka řádky předem omezena, může se stát, že pro ni nebude k dispozici dostatečně velká vyrovnávací paměť. Také to ale nemusí být vždy žádoucí - někdy může být vhodné, či dokonce nutné odesílat menší celky než celé řádky, až např. po jednotlivé znaky. S touto možností protokol TELNET počítá, a doporučuje ji realizovat. Přesný mechanismus, kterým by si zdroj dat mohl vynutit jejich odeslání ještě před zakončením řádky, je však ponechán na konkrétní implementaci.

Ostatní je na vzájemné dohodě

Pro správné pochopení smyslu a role virtuálního terminálu NVT je velmi důležité si uvědomit to, co jsme si již naznačili v 66. dílu - že totiž každý virtuální terminál vždy omezuje "individualitu" konkrétních terminálů, a redukuje jejich vlastnosti a schopnosti na takovou úroveň, která může být společná prakticky všem fyzicky existujícím terminálům. Nejinak je tomu i v případě virtuálního terminálu NVT, který si proto můžeme představit jako největší společný jmenovatel všech ještě použitelných terminálů. Protokol TELNET jej však chápe jen jako "povinné minimum" a připouští, aby se obě strany mohly v konkrétním případě dohodnout "na lepším" - tedy na tom, že mají a jsou schopny používat nějaká rozšíření vůči tomu, co požaduje NVT.

Přitom ale platí zásada, že použití rozšíření (anglicky: **options**) si nelze vynucovat - každá strana má právo vznášet návrhy na jejich použití, ale druhá strana má vždy právo je odmítnout. Díky tomu je možné vystačit i s "hloupými" terminály, jejichž skutečné schopnosti nepřesahují minimum, požadované virtuálním terminálem NVT, a na druhé straně je možné efektivně využít možnosti a schopnosti lépe vybavených terminálů.

Bezprostředně po navázání spojení tedy obě strany mohou používat právě a pouze to, co jim zaručuje virtuální terminál NVT. Mohou však kdykoli zahájit "licitaci", v rámci které se dohodnou na použití oboustranně přijatelných rozšíření. Vzájemné domlouvání na použití různých rozšíření obvykle probíhá okamžitě po navázání spojení, ale není to nutnou podmínkou. Stejně tak se mohou obě strany kdykoli dohodnout na tom, že přestanou určité rozšíření používat. Zajímavá je v tomto ohledu také zásada rovnoprávnosti - každá ze stran má stejné právo navrhnout používání

určitého rozšíření, a stejně tak může požadovat i ukončení používání určitého rozšíření (žádosti tohoto typu přitom nesmí být druhou stranou odmítnuty).

Protokol TELNET samozřejmě musí definovat konkrétní způsob, jakým mají obě strany postupovat při vzájemné "licitaci" (anglicky: options negotiation). Příslušný mechanismus je ovšem koncipován jako otevřený - nemůže totiž anticipovat všechna budoucí rozšíření, která budou připadat v úvahu, a tak pro ně nemůže přesně předepisovat konkrétní formu "licitace". Místo toho ponechává otevřený prostor pro individuální postupy vzájemného dohadování, ale současně s tím je umožňuje jednoznačně detekovat, tak aby druhá strana měla vždy možnost rozpoznat, že jde o nabídku používání rozšíření, a i když jí nerozumí, mohla ji odmítnout.

Sedmibitové znaky v osmibitových bytech

Pro kódování jednotlivých znaků používá protokol TELNET znakový kód ASCII. Požaduje, aby tiskárna (resp. zobrazovací zařízení) virtuálního terminálu NVT byla schopna znázornit všech 95 alfanumerických znaků ASCII (s kódy 32 až 127), a z 33 řídících znaků povinně vyžaduje interpretaci znaků NULL, CR a LF. Kromě toho stanovuje přesný význam i pro řídící znaky BEL (Bell), BS (Back Space), HT (Horizontal Tab), VT (Vertical Tab) a FF (Form Feed), a to v souladu s jejich významem v kódu ASCII - ovšem s tím, že interpretace těchto znaků není povinná (tj. tiskárna je nemusí interpretovat vůbec, ale pokud ano, pak jen stanoveným způsobem). Pro ostatní řídící znaky kódu ASCII je stanoveno, že nebudou mít na tiskárnu žádný efekt.

Po klávesnici virtuálního terminálu NVT je naopak požadováno, aby byla schopna generovat všech 128 znaků kódu ASCII (i když některé nemají na tiskárnu žádný efekt). Kromě toho je požadováno, aby klávesnice NVT generovala ještě i několik dalších znaků, které mají význam řídících příkazů protokolu TELNET (o nich bude řeč příště).

Jednotlivé znaky sedmibitového kódu ASCII jsou ovšem přenášeny zásadně v osmi bitech. Díky tomu je pak možné k nim "přidat" ještě i právě naznačené řídící příkazy (odlišené nastavením nejvyššího bitu). Jako jedno z možných rozšření se ale obě strany mohou dohodnout na tom, že si budou předávat osmibitové znaky (tj. znaky, kódované v osmi bitech). Pak je ovšem nutné zajistit potřebnou transparenci - umožňující jednoznačně odlišit příkazy od "užitečných dat" - jiným způsobem.

69/ TELNET - III

V minulém dílu jsme se zabývali virtuálním terminálem NVT protokolu TELNET. Dospěli jsme k představě, že jde jen o jisté "povinné minimum", jehož možnosti a schopnosti si mohou obě strany na základě vzájemné dohody rozšířit. Dnes se již dostaneme ke konkrétním mechanismům, které se přitom používají.

Na úvod je vhodné si znovu zdůraznit, že veškerá komunikace mezi klientskou a serverovou složkou protokolu TELNET má zásadně charakter přenosu znaků. Mají-li pak být v takovémto prostředí implementovány jakékoli řídící mechanismy a postupy, musí mít příslušné řídící příkazy i veškeré reakce na ně opět povahu znaků. Musí však být zajištěna také potřebná transparence dat - musí být možné jednoznačně rozlišit, které znaky představují "užitečná data" a které naopak představují řídící příkazy.

Tvůrci protokolu TELNET měli v zásadě dvě možnosti, jak tohoto cíle dosáhnout. Jednou bylo přisoudit význam řídících příkazů některým ASCII znakům. Tím by se ale připravili o možnost používat tytéž znaky v jejich původním významu, který mají v kódu ASCII, a navíc by si tím dosti omezili možný repertoár vlastních řídících příkazů (protože takto využitelných znaků ASCII je jen velmi omezený počet). Proto se autoři protokolu TELNET raději rozhodli pro druhou možnost, kterou představuje samostatné kódování řídících příkazů, nezávislé na kódu ASCII.

Zajištění transparence

Jak jsme si již uvedli minule, protokol TELNET přenáší jednotlivé znaky v osmibitových bytech. Znaky kódu ASCII jsou však jen sedmibitové, a pro potřeby přenosu se doplňují nejvyšším bitem, nastaveným na nulu. Pro kódování svých řídících příkazů se pak tvůrcům protokolu TELNET nabízelo všech 128 možných 8-bitových hodnot, s nejvyšším bitem nastaveným na jedničku - a tuto možnost také skutečně využili. Zde je ovšem důležité si uvědomit, že virtuální terminál NVT sice předpokládá pouze přenos sedmibitových ASCII znaků, ale že jednou z možností rozšíření je přenos osmibitových znaků. Jakmile se ale toto rozšíření použije, okamžitě se tím ztrácí automatické rozlišení mezi "užitečnými" znaky a řídícími příkazy podle hodnoty nejvyššího (přesněji: nejvýznamnějšího) bitu.

Samotný způsob kódování řídících příkazů tedy ještě nemohl zajistit potřebnou transparenci dat, a proto tvůrcům protokolu TELNET nezbylo než použít řešení, obvyklé např. u znakově orientovaných linkových protokolů či u řídících jazyků pro ovládání tiskáren a obdobných znakově orientovaných zařízení: před samotný řídící znak zařadit speciální znak, který změní interpretaci jednoho, resp. několika následujících znaků. V případě tiskáren jde o znak ESCAPE (s ASCII kódem 27, resp. 1B hexadecimálně), který uvozuje tzv. ESCAPE sekvence, zatímco u linkových protokolů jde o znak DLE (Data Link ESCAPE, s ASCII kódem 16, resp. 10 hexadecimálně), který prefixuje řídící znaky začátku a konce rámce apod. V případě protokolu TELNET byl zvolen znak s číselných kódem 255 (resp. FF hexadecimálně), označovaný jako **IAC** (Interpret As Command, dolova: interpretuj jako řídící příkaz), který povinně uvozuje všechny řídící příkazy. Pokud by se číselný kód tohoto znaku (tj. kód 255) vyskytl v "užitečných" datech, musí být zdvojen, čím se zamezí jeho interpretaci jako uvozujícího znaku řídícího příkazu.

Příkazy protokolu TELNET

Většina řídících příkazů protokolu TELNET je reprezentována jedním řídícím znakem, ke kterému se přidává povinný prefix - znak IAC - takže nekratší příkaz je tvořen dvěma osmibitovými byty. Existují ovšem i příkazy, pro jejichž vyjádření jediný řídící znak nestačí, a je nutné použít ještě jeden dodatečný znak. I v tomto případě se ale používá jediný uvozující znak IAC, takže výsledný příkaz je tvořen třemi byty. Pro ještě "delší" příkazy se pak již používá poněkud odlišný mechanismus, se kterým se seznámíme posléze.

Podle svého významu spadají řídící příkazy protokolu TELNET do tří skupin, mezi:

- příkazy pro editaci (jsou jen dva: pro vymazání aktuální řádky a pro vymazání předchozího znaku),

- příkazy pro řízení komunikace mezi oběma stranami (například pro přerušení aplikačního procesu na straně serveru, pro zastavení jeho výstupu, pro vzájemnou synchronizaci apod.),

- příkazy, umožňující oběma stranám dohodnout se na použití konkrétních rozšíření.

Na poslední skupinu se nyní zaměříme podrobněji.

Licitace o rozšíření

Jak jsme si již naznačili minule, použití nejrůznějších rozšíření oproti virtuálnímu terminálu NVT je nepovinné. Každá ze stran má právo navrhnout použití takového rozšíření, jaké uzná za vhodné, ale druhá strana má plné právo jej odmítnout. Výrazný prvek demokratičnosti je pak i v tom, že právo navrhovat použití rozšření má nejen složka v roli klienta, ale i složka v roli serveru - obě strany mají v tomto ohledu rovnoprávné postavení.

Zajímavý je i konkrétní způsob vzájemné "licitace". Ta strana, která chce iniciovat použití určitého rozšíření, má dvě možnosti: navrhnout, že sama přijme nějaké opatření (tj. sama začne používat příslušné rozšíření), a ptát se druhé strany, zda s tím souhlasí, nebo naopak navrhnout druhé straně, aby ona přijala určité opatření. Ukažme si na příkladu, v čem může spočívat rozdíl, a proč je vůbec nutné jej uvažovat: virtuální terminál NVT předpokládá, že každá strana bude používat tzv. lokální echo (local echo) - tedy že veškeré znaky, zadané z klávesnice, si bude sama ihned zobrazovat na svém displeji (resp. tisknout na tiskárně, kterou předpokládá NVT), a současně s tím je odesílat druhé straně. Díky tomu uživatel okamžitě vidí, co vlastně z klávesnice zadal, a co se druhé straně odeslalo. Alternativa je taková, že zadaný znak je pouze odeslán druhé straně, a teprve ta jej zase pošle zpět k zobrazení na displeji uživatelova terminálu. Používání této techniky, označované jako tzv. vzdálené echo (remote echo), je jedním z možných rozšření virtuálního terminálu NVT. Je ovšem zásadní rozdíl mezi tím, kdy jedna strana navrhne druhé, že ona bude provádět vzdálené echo (tj. že bude posílat zpět každý znak, který od druhé strany dostane), a kdy vzdálené echo požaduje na druhé straně.

Některá rozšíření virtuálního terminálu NVT, jako právě naznačené používání tzv. vzdáleného echa, mají asymetrický charakter, a mohou být používány oběma stranami současně, žádnou z nich, nebo jen jednou ze zúčastněných stran. Naproti tomu jiná rozšíření mají symetrický charakter, a musí je používat buď obě strany současně, nebo žádná.

Protokol TELNET nabízí celkem čtyři příkazy, pomocí kterých mohou obě strany "licitovat" o použití rozšíření. Jsou to po příkazy:

WILL (doslova: budu), vysílající strana tímto příkazem signalizuje, že chce sama začít používat určité rozšíření, a dotazuje se druhé strany na její souhlas

DO (doslova: dělej), vysílající strana tímto příkazem vyzývá druhou stranu, aby ona začala používat určité opatření

WON'T (od: Will Not, doslova: nebudu), tímto příkazem vysílající strana odmítá výzvu druhé strany (vyjádřenou příkazem DO), případně signalizuje, že sama již nebude dále pokračovat v používání určitého rozšření (pokud jej dosud používala)

DON'T (od: Do Not, doslova: nedělej), tímto příkazem vysílající strana odmítá nabídku druhé strany (vyjádřenou příkazem WILL), případně signalizuje, že od druhé strany již neočekává, že bude nadále používat určité rozšíření.

Příkazy WON'T a DON'T jsou tedy zápornými odpověďmi na výzvu, vyjádřenou po řadě příkazy DO a WILL. Jaké jsou ale kladné odpovědi? Kladnou odpovědí na příkaz DO (výzvu "dělej") je příkaz WILL (tj. "budu"), a naopak kladnou odpovědí na příkaz WILL (nabídku "budu") je příkaz DO (tj. "dělej").

Identifikace rozšíření

Příkazy WILL, DO, WON'T a DON'T samy o sobě ještě neurčují jednu velmi podstatnou věc - o jaké konkrétní rozšíření se jedná. Toto je určeno až dalším rozlišujícím parametrem (ve skutečnosti číselným kódem v rozsahu jednoho osmibitového bytu), který následuje bezprostředně za vlastními příkazy. Takže například nabídka jedné strany, že bude druhé straně zajišťovat vzdálené echo, se symbolicky zapisuje jako

IAC WILL ECHO

a ve skutečnosti se vysílá jako trojice bytů s hodnotami po řadě 255, 251 a 1 (kde 255 je kód uvozujícího znaku IAC, 251 je kód příkazu WILL, a 1 je kód rozšíření o vzdálené echo). Druhá strana v případě souhlasu odpoví sekvencí

IAC DO ECHO

(číselně: 255 253 1), a v případě nesouhlasu sekvencí

IAC DON'T ECHO

(číselně: 255 254 1).

Obdobně požadavek jedné strany, aby pro ni druhá strana zajišťovala vzdálené echo, by měl tvar

IAC DO ECHO

(číselně: 255 253 1). Druhá strana akceptuje tuto výzvu sekvencí

IAC WILL ECHO

(číselně: 255 251 1), resp. odmítá ji pomocí

IAC WON'T ECHO

(číselně: 255 252 1). Aby se zabránilo možnému zacyklení, kdyby jedna strana považovala odpověď druhé strany za novou žádost, je zavedeno následující pravidlo: na žádost o použití takového rozšíření, které již je používáno, se neodpovídá nijak.

Počet a význam jednotlivých rozšíření není předem stanoven. Nová rozšíření mohou vznikat a skutečně vznikají na základě skutečných potřeb a vývoje výpočetní techniky. Pro jejich koordinaci však musí existovat jediná centrální autorita, která jim přiděluje jejich číselné kódy. Tou je pan John Postel z University of Southern California, autor mnoha standardů Internetu, vydávaných ve formě tzv. RFC dokumentů.

Podrobnější licitace

Z některých zajímavých rozšíření (které uvádí tabulka 69.1) stojí za zmínku např. možnost dohodnout se na použití konkrétního typu terminálu (s větším repertoárem schopností a možností, než jaké nabízí "základní NVT"). V současné době, kdy je velká část terminálových relací zajišťována prostřednictvím terminálové emulace (viz 65. díl), a příslušné emulátory jsou často schopny emulovat více různých terminálů, je dokonce žádoucí, aby se obě strany mohly dohodnout na použití nejvhodnějšího terminálu.

Pro tyto účely však již není výše naznačený mechanismus dostatečný. Umožňuje pouze to, aby se obě strany dohodly na tom, že se vůbec chtějí (a umějí) domlouvat na typu terminálu. Poté musí následovat druhá fáze "licitace" (v angličtině označovaná jako **subnegotiation**), v rámci které se již mohou podrobněji domlouvat na konkrétních parametrech. V případě volby typu terminálu by v rámci této druhé fáze složka v roli klienta předkládala složce v roli serveru jednotlivé typy terminálů, které je schopna emulovat, a server by si z nich vybíral tu, kterou považuje ze svého pohledu za nejvýhodnější.

Obdobná úvaha platí i pro jiná rozšíření: například pro stanovení velikosti displeje, kde se konkrétní počet řádek na stránce a počet znaků na řádce sděluje opět v druhé fázi (subnegotiation) "licitace".

Konkrétní forma této druhé fáze není předepsána, neboť je určena až potřebami příslušného rozšíření. Závazný je pouze způsob zajištění transparence: posloupnost znaků, předávaných v rámci druhé fáze "licitace", musí být uvozena řídícím znakem SB (Start of Subnegotiation), a zakončena znakem SE (End of Subnegotiation). Oba tyto znaky přitom musí být samy prefixovány znakem IAC.

Symb. označení číselný kód význam

TRANSMIT-BINARY 0 přenos 8-bitových znaků (resp. binárních dat)

ECHO 1 tzv. vzdálené echo (viz text)

STATUS 5 toto rozšíření umožňuje, aby jedna strana vysílala druhé straně informace o svému stavu (tzv. status), resp. aby si jedna strana vyžádala od druhé informace o jejím stavu

TERMINAL-TYPE 24 díky tomuto rozšíření se zúčastněné strany mohou dohodnout na typu používaného terminálu

NAWS 31 (Negotiate About Window Size), toto rozšíření umožňuje zúčastněným stranám dohodnout se velikosti displeje terminálu

TERMINAL-SPEED 32 obdobně, pro přenosovou rychlost terminálu

TOGGLE-FLOW-CONTROL 33 toto rozšíření umožňuje zúčastněným stranám dynamicky měnit způsob řízení toku

EXTENDED-OPTIONS-LIST 255 toto rozšíření je ve skutečnosti mechanismem, pomocí kterého lze zvýšit počet možných rozšíření nad 255

Tabulka 69.1: Příklady rozšíření virtuálního terminálu NVT protokolu TELNET

70/ Přenos a sdílení souborů

V posledních pěti dílech tohoto seriálu jsme se podrobněji zabývali první z aplikačních služeb v rámci síťového modelu TCP/IP - vzdáleným přihlašováním. Dnes již přijde na řadu další velmi potřebná služba: přenos a sdílení souborů. Stejně jako v předchozím případě se ale nejprve zamyslíme nad tím, o co vlastně jde, a teprve pak si ukážeme, jak je přenos souborů a jejich sdílení konkrétně řešeno v TCP/IP sítích.

Problematika přenosu a sdílení souborů v počítačových sítích má společný základ v situaci, kterou je možné lapidárně vyjádřit slovy: "chceme pracovat se souborem, který se nachází na jiném uzlovém počítači". Buďme ale přesnější: obsah souboru, který se nachází na jiném uzlu, chceme zpracovávat na našem počítači. Nejde nám tedy o takové řešení, kdy bychom se prostřednictvím vzdáleného přihlašování dostali se svým počítačem do role terminálu toho uzlu, na kterém se příslušný soubor nachází, a zde jej zpracovávali jako "místní" soubor. Představme si například, že chceme editovat nějaký soubor - pak by tato druhá varianta odpovídala situaci, kdy editor běží jako aplikační program na stejném počítači, na kterém se nachází i editovaný soubor. Nám půjde o případ, kdy chceme používat editor běžící na našem počítači, a jeho prostřednictvím editovat obsah souboru, který se nachází na vzdáleném počítači.

Vícenásobný přístup

Řešení právě naznačeného problému má přinejmenším dvě stránky. Tou první je přenos celého souboru či jeho částí z místa, kde se nachází, na místo kde má být zpracován, a ev. zpět. Druhou stránkou je pak otázka vícenásobného přístupu k jednomu a témuž souboru - tedy řešení situace, kdy se o přístup k určitému souboru uchází více zájemců současně.

Vícenásobný přístup samozřejmě není specialitou počítačových sítí, přesně stejný problém totiž nastává obecně u každého víceuživatelského, a dokonce i jen víceúlohového systému. Jeho řešení v prostředí počítačových sítí se nemusí nijak principiálně lišit od řešení na izolovaných počítačích, je spíše jen ztíženo existencí přenosových cest s omezenou kapacitou a zpožděními při přenosu.

Pro zajištění korektního vícenásobného přístupu k souborům existuje celá řada technik, od jednoduchého uzamykání celých souborů či jejich jednotlivých částí až po velmi propracované metody, které mají svůj původ v prostředí distribuovaných systémů.

V této části seriálu se však problematikou vícenásobného přístupu zabývat nebudeme, a zaměříme se hlavně na vlastní přenos souborů mezi uzlovými počítači sítě.

Transparentní vs. netransparentní přístup

Potřebujeme-li na jednom počítači zpracovávat obsah souboru, který se nachází na jiném uzlovém počítači, musí vždy dojít k přenosu tohoto souboru (či alespoň jeho části) tam, kde má být zpracován. Existují ale dva zásadně odlišné přístupy k tomu,

kdo a jak tento přenos vyvolává: jeden přístup můžeme označit jako **transparentní**, a druhý jako **netransparentní**.

Netransparentní přístup je založen na myšlence, že přenos souborů bude ponechán plně na koncovém uživateli. Když chce pracovat s nějakým souborem, musí si jej sám a explicitně (pomocí vhodné utility, kterou si za tímto účelem vyvolá) nejprve přenést na svůj počítač, a zde jej pak zpracovat stejným způsobem, jako kterýkoli jiný "místní" soubor (a pak jej zase sám přenést zpět, pokud je to zapotřebí). V tomto případě si tedy koncový uživatel plně uvědomuje, že různé soubory se nachází na různých počítačích (dokonce musí přesně vědět kde), a tudíž pro něj existuje principiální rozdíl mezi "místními" a "vzdálenými" soubory.

Alternativou je transparentní přístup k souborům. Ten se snaží vytvářet iluzi, že vzdálené soubory se nachází na místním počítači, snaží se zakrýt veškeré rozdíly mezi oběma druhy souborů, a přebírá na sebe veškerou agendu, která je s tímto předstíráním spojena. Pro koncového uživatele (i pro jím spouštěnou aplikaci) se pak i vzdálené soubory "tváří" naprosto stejně, jako soubory místní, a stejně se s nimi i pracuje. Uživatel a jeho aplikace si pak vlastně vůbec nemusí uvědomovat existenci počítačové sítě a jakkoli distribuovaného prostředí, a tudíž ani to, že různé soubory se mohou ve skutečnosti nacházet na různých počítačích.

Zajištění plně transparentního přístupu k souborům je samozřejmě mnohem náročnější, než zajištění přístupu netransparentního. Součástí síťového modelu TCP/IP jsou protokoly pro oba tyto přístupy (protokoly FTP a NFS), a my se budeme oběma z nich podrobněji zabývat.

Ještě dříve si však alespoň naznačme, s jakými principiálními problémy je realizace obou základních přístupů spojena. Nejvíce jich samozřejmě bude u transparentního přístupu, jehož implementace je nejobtížnější - musí být totiž skryta pod operačním systémem, který nezbytný přenos aktivuje v okamžiku, kdy je uživatelem či aplikací požádán o zajištění přístupu k určitému souboru, a zjistí že tento ve skutečnosti není místní.

Problém je v rozdílnosti operačních systémů

Hlavní zdroj problémů před námi vyvstane v okamžiku, kdy si uvědomíme, že přístup k souborům v počítačové síti může být zajišťován i z prostředí různých operačních systémů. Například je možné (a při dnešní popularitě osobních počítačů a lokálních sítí i docela běžné) aby uživatel počítače PC s operačním systémem MS DOS měl přístup k souborům, které se ve skutečnosti nachází na jiném uzlovém počítači, na kterém je provozován například operační systém Unix. Ukažme si dvě hlavní oblasti, ve kterých se pohled obou operačních systémů na soubory výrazně liší:

- jména a přípony; MS DOS připouští maximálně osmiznaková jména souborů (a tříznakové přípony), zatímco Unix dovoluje používat je jménech i příponách znaků více (konkrétní počet závisí na variantě Unixu), a navíc to mohou být i některé speciální znaky, které naopak v DOSovských jménech a příponách přípustné nejsou. Další rozdíl je v tom, že MS DOS nerozlišuje velká a malá písmena, zatímco Unix ano. Obecně proto platí, že každé jméno a přípona, přípustné v MS DOSu, je přípustné i v Unixu, ale naopak nikoli. Při přístupu k Unixovským souborům z MS DOSu je proto nutné vhodně nahradit případná nepřípustná jména. Při netransparentním přístupu je možné vyřešit celý problém vcelku snadno - zde si totiž

uživatel musí zavolat příslušnou utilitu, a jejím prostřednictvím explicitně zadávat, které soubory chce přenést. Pak není problém vyžádat si na něm i jméno (přípustné pro DOS), pod jakým má být Unixovský soubor přenesen do prostředí MS DOSu. Při plně transparentním přístupu však takovéto řešení není možné, a tak musí být vhodně implementována automatická transformace Unixovských jmen souborů na DOSsovská (což může být velkým oříškem - jak zajistit jednoznačnost, neboli to, aby se dvě různá Unixovská jména nepromítla do stejného DOSovského jména).

- vlastnictví souborů a přístupová práva; MS DOS je jednouživatelský operační systém. Předpokládá, že s ním bude vždy pracovat jen jeden uživatel, kterému také budou patřit všechny soubory. Proto nemá zapotřebí uvažovat, že by různé soubory mohly mít různé majitele, a ti mohli přístup ke svým soborům jiným uživatelům nějak omezovat - z tohoto pohledu je příznačné, že MS DOS pojem uživatele vlastně ani nezná. Naproti tomu Unix je víceuživatelským systémem, který tudíž jednotlivé uživatele rozlišovat musí, a u každého souboru předpokládá, že někomu patří (tj. že každý soubor má svého majitele). Majitel souboru pak může definovat přístupová práva jiných uživatelů k tomuto souboru (a sobě samotnému také). Aby tak nemusel činit pro každého individuálního uživatele samostatně, jsou v Unixu zavedeny skupiny uživatelů, a každý uživatel vždy do nějaké skupiny patří. Ke každému souboru se pak explicitně definují přístupová práva pro vlastníka souboru, pro skupinu, do které vlastník patří, a pro ostatní uživatele. Takto je možné samostatně nastavit právo ke čtení souboru, právo k zápisu do něj (resp. přepisu), a právo ke spuštění (provedení spustitelného programu).

Má-li však být Unixovský soubor přenesen do prostředí MS DOSu nebo naopak, podle čeho se bude kontrolovat oprávněnost přístupu k tomuto souboru? Ten, kdo z MS DOSu vznáší svůj požadavek na přístup k souboru, je vlastně anonymní (když MS DOS pojem uživatele nezná). V případě netransparentního přístupu ze strany MS DOSu je řešení opět vcelku jednoduché - příslušná utilita (aplikační program) na straně DOSU se může svého uživatele vyptat, jménem koho má vznést svůj požadavek na straně Unixu (tj. vyžádá si Unixovské uživatelské jméno). Podle tohoto jména se pak posuzuje oprávněnost přístup k příslušnému souboru. Jak to ale udělat v případě plně transparentního přístupu? Zde připadá v úvahu v podstatě jen jediná možnost: Unixovské uživatelské jméno (a heslo) si vyžádat předem, a všechny následné požadavky (které již jsou generovány bez přímého vědomí uživatele), vznášet tímto jménem.

Jak se to konkrétně dělá, k tomu se včas dostaneme.

71/ FTP - I.

V minulém dílu jsme si naznačili, že přenos a sdílení souborů v počítačových sítích může být řešeno dvěma principiálně odlišnými způsoby: transparentně a netransparentně. Dnes se již dostaneme k tomu, jak je v síťovém modelu TCP/IP zajišťován netransparentní přenos - tedy k protokolu FTP.

Nejprve si ale znovu zopakujme, v čem je rozdíl mezi transparentním a netransparentním řešením: v případě netransparentního přístupu si uživatel plně uvědomuje rozdíl mezi místními soubory a soubory na vzdáleném počítači, a musí se sám a explicitně starat o jejich přenos, zatímco v případě netransparentního přístupu místní a vzdálené soubory splývají, pro uživatele mezi nimi není žádný principiální

rozdíl, a veškerou agendu spojenou s předstíráním této iluze na sebe přebírá síťový software a operační systém.

Minule jsme se již také zmínili o tom, že síťový model TCP/IP pamatuje na obě varianty. Pro netransparentní variantu nabízí hned dva protokoly - protokol **FTP** (**File Transfer Protocol**, doslova: protokol pro přenos souborů), který lze označit jako "plnohodnotný", vybavený množstvím různých schopností a funkcí, a dále mnohem jednodušší protokol **TFTP** (**Trivial File Transfer Protocol**), poskytující jen základní funkce, ale zato implementačně jednodušší, a často i rychlejší. Naše povídání začneme filosofií protokolu FTP.

FTP je starší než TCP/IP

Na protokolu FTP je dobře patrný jeden význačný rys celého síťového modelu TCP/IP - totiž skutečnost, že nevznikl naráz, od zeleného stolu, ale postupným vývojem a zdokonalováním. Jak jsme si uvedli již ve 42. dílu tohoto seriálu, protokoly TCP/IP získaly svou dnešní podobu zhruba v letech 1977 až 1979, a do sítě ARPANET se začaly prosazovat přibližně od roku 1980, kdy se z ARPANETu postupně stává zárodek dnešního Internetu. První verze protokolu FTP pro netransparentní přenos souborů však pochází již z roku 1971, a od té doby prošla dlouhým vývojem, podporovaným širokou diskusí uživatelské veřejnosti. Od začátku osmdesátých let, kdy i protokol FTP přešel na používání transportního protokolu TCP (místo původního transportního protokolu NCP sítě ARPANET) se ale již zásadněji neměnil.

Správné zasazení protokolu FTP do historického rámce nám pomůže snáze pochopit některé jeho koncepční rysy.

Jak se FTP vyrovnává s různorodostí

Již minule jsme si naznačili, že přenos a sdílení souborů mezi různými uzlovými počítači může narážet na mnohé problémy. Ukázali jsme si některé z nich, které vyplývají z odlišného pohledu různých operačních systémů na soubory, a to na "novodobém" příkladu Unixu a MS DOSu. V době, kdy koncepce protokolu FTP vznikala a utvářela se, však existovaly ještě mnohem základnější rozdíly mezi jednotlivými počítači - některé z nich používaly osmibitové byty a 32-bitová slova, zatímco jiné 36-bytová slova. Některé pracovaly s textem tak, že jeho jednotlivé znaky ukládaly do 8-bitových bytů v kódu EBCDIC (zejména střediskové počítače IBM) nebo v kódu ASCII, zatímco jiné ukládaly znaky po čtveřicích do 36-bitových slov (každý znak do 9 bitů), a ještě jiné jich do 36-bitových slov "nacpaly" až pět (každý znak do sedmi bitů). Některé počítače přitom umisťovaly jednotlivé řádky textu do záznamů (records) pevné délky, zatímco jiné chápaly text jako lineární posloupnost jednotlivých znaků, členěnou vloženými znaky pro přechod na novou řádku.

Protokol FTP se vyrovnává se všemi těmito odlišnostmi v podstatě jediným možným způsobem: zavádí jednotný tvar, ve kterém jsou data skutečně přenášena, a ponechává na obou koncových účastnících, aby zajistili veškeré potřebné převody z/do místních konvencí.

Zároveň ale protokol FTP vychází vstříc i případům, kdy by jediný a neměnný přenosový tvar byl neefektivní, a umožňuje jej v určitém rozmezí a po dohodě obou

stran měnit. V podstatě lze říci, že tento přenosový tvar má tři stupně volnosti: režim přenosu, strukturu souborů a reprezentaci dat.

Reprezentace dat

Při vlastním přenosu jsou veškerá data přenášena zásadně jako 8-bitové byty. Protokol FTP však počítá s tím, že počítače, na kterých jsou tato data uchovávána, mohou používat jiné konvence - například 36-bitová slova, jiný způsob ukládání jednotlivých znaků do celých slov (viz výše), jiný znakový kód než ASCII, apod. Z tohoto důvodu musí být na straně příjemce a/nebo odesilatele zajišťovány potřebné konverze. Jejich provádění ovšem vyžaduje, aby obě strany věděly, co si vlastně mezi sebou přenáší: zda jde o text, složený z jednotlivých znaků, nebo zda jde o binární data, která mají být chápána jako lineární posloupnost bitů, nebo zde jde například o přenos jednotlivých bytů nestandardní velikosti (např. devítibitových bytů).

Protokol FTP implicitně předpokládá, že přenášená data představují ASCII text. Ten se pak skutečně přenáší v tom formátu, který pro ASCII text požaduje protokol Telnet (viz 68. díl). Jednotlivé znaky jsou tedy přenášeny jako 8-bitové, a řádky jsou oddělovány dvojicí znaků CRLF. Odesilatel proto musí převádět jednotlivé znaky z takového kódu a způsobu zobrazení, jaký používá jeho počítač, na právě popsaný přenosový tvar (tj. převádět je do kódu ASCII, ukládat po jednom do osmi bitů, a přechody na nové řádky reprezentovat dvojicí znaků CR a LF). Analogicky se musí chovat i příjemce.

Protokol FTP však umožňuje, aby se obě strany dohodly na jiném významu přenášených dat. Možnosti jsou následující:

- text v kódu EBCDIC (přenášená data jsou tvořena jednotlivými znaky, tyto jsou opět znázorněny v 8 bitech, ale pro jejich kódování je místo znakového kódu ASCII použit znakový kód EBCDIC). Tato varianta vychází vstříc přenosům textů mezi střediskovými počítači IBM, které používají právě znakový kód EBCDIC, a eliminuje tak dvojí konverzi každého znaku.

- binární data (tzv. image type). V tomto případě obě strany chápou přenášená data jako spojitou posloupnost bitů, které jsou členěny na 8-bitové byty jen pro potřeby vlastního přenosu.

- byty nestandardní velikosti (tzv. local byte). V tomto případě může odesilatel stanovit, jak velké byty sám používá, a dát příjemci možnost se tomu přizpůsobit. Například počítač, který pracuje s 36-bitovými slovy, může na tuto skutečnost upozornit příjemce, a ten pak může například ukládat jednotlivá 36-bitová slova do dvou 32-bitových slov apod. Zde je ovšem třeba si uvědomit, že jde jen o "logickou" velikost bytu, která slouží oběma stranám jako vodítko pro provádění potřebných konverzí. Nastavení nestandardní velikosti bytu nemá žádný vliv na skutečnost, že vlastní data jsou doopravdy přenášena v 8-bitových bytech.

Struktura souborů

Další odlišností, se kterou se protokol FTP musel vyrovnat, je rozdílný pohled na to, jak je soubor vnitřně členěn. Zda je například považován za lineární posloupnost bytů, nebo je členěn na sekvenční záznamy (records), nebo zda je tvořen navzájem nezávislými stránkami, které jsou opatřeny indexy, a mohou vytvářet nespojitý soubor.

V této souvislosti je opět třeba mít na paměti, že různé systémy mohou pro jeden a tentýž druh souboru používat jinou vnitřní strukturu - například texty se na většině počítačů uchovávají v textových souborech, tvořených lineární posloupností jednotlivých znaků (bytů), ale například na střediskových počítačích IBM jsou textové soubory organizovány jako posloupnosti záznamů pevné délky. I zde pak musí nastoupit vhodné konverze na jedné straně či na obou současně, ale opět platí zásada, že každý musí vědět, co se vlastně přenáší.

Pokud není řečeno jinak, protokol FTP implicitně předpokládá, že přenášený soubor nemá žádnou vnitřní strukturu, a je tvořen lineární posloupností jednotlivých datových bytů (tato možnost je označována jako file structure). Další možnosti, na kterých se mohou zúčastněné strany dohodnout, je soubor členěný na sekvenční záznamy (record structure) a soubor, členěný na stránky (page structure).

Režim přenosu

Třetím stupněm volnosti je režim přenosu dat. Důvodem pro zavedení více různých režimů byla zřejmě snaha vyrovnat se s nedokonalostí a omezenou kapacitou přenosových cest. Protokol FTP totiž umožňuje provádět i určitou kompresi přenášených dat, a také zotavit se z výpadku spojení v průběhu přenosu a po obnovení spojení v něm pokračovat.

Implicitní režim přenosu (označovaný jako stream mode) je takový, kdy přenášená data jsou chápána jen jako spojitý proud (stream) bytů, a jako taková jsou také přenášena.

Je ovšem možné zvolit i jiný režim (blokový režim, block mode), při kterém jsou přenášená data členěna do bloků, a každý z nich je opatřen hlavičkou (která obsahuje mj. údaj o délce bloku, příznak posledního bloku apod.). V tomto režimu je pak možné vkládat mezi bloky, obsahující vlastní data, i malé speciální bloky (identifikované příznakem ve své hlavičce), které ve skutečnosti slouží jako záložky (kontrolní body). Po výpadku spojení a jeho následném obnovení je pak možné využít existence těchto značek k indikaci místa, odkud má být přerušený přenos opakován.

Třetím možným režimem přenosu je tzv. zhuštěný režim (compressed mode), při kterém jsou přenášená data komprimována. Nejde však o žádnou propracovanou a efektivní kompresní techniku, jaké známe ze současné doby, ale jen o pouhou eliminaci znaků, které se opakují několikrát za sebou (typickým příkladem jsou posloupnosti mezer v textových souborech). Takováto posloupnost stejných znaků je pro potřeby přenosu nahrazena pouze jedním exemplářem příslušného znaku a údajem o tom, kolikrát se v originále opakuje.

72/ FTP - II.

V předchozím dílu jsme se začali zabývat konkrétními vlastnostmi protokolu FTP, který má v rodině protokolů TCP/IP na starosti netransparentní přenos souborů. Přitom jsme se věnovali hlavně způsobu, jakým se tento protokol vyrovnává s odlišnostmi různých počítačů a jejich operačních systémů v pohledu na soubory, jejich vnitřní strukturu a vlastní obsah. Dnes se již dostaneme k tomu, jak přenos souborů podle protokolu FTP v praxi probíhá - kým je iniciován, kým je řízen, jakými příkazy atd.

Implementace protokolu FTP vychází z architektury klient-server, kterou jsme si přiblížili již v 64. dílu seriálu. Předpokládá tedy existenci dvou navzájem nerovnoprávných složek: klienta a serveru. Složka v roli klienta je iniciátorem, z jehož popudu k přenosům souborů dochází, a server je druhou stranou, která přitom spolupracuje. Server tedy nabízí jako svou službu spolupráci na přenosu souborů, a klient je tím, kdo o tuto službu žádá. Typicky je složka v roli klienta představována aplikačním programem, který si uživatel spouští na svém počítači, a složka v roli serveru démonem (FTP démonem v prostředí Unixu, resp. obdobným systémovým procesem v jiném prostředí) na tom počítači, ze kterého či na který uživatel chce přenést nějaký soubor. Vlastní směr přenosu (od serveru ke klientovi či naopak) přitom není pro rozlišení obou složek relevantní.

Trvale jen nezbytné minimum

Protokol FTP vychází vstříc takovému způsobu implementace, který se snaží minimalizovat své nároky na různé systémové prostředky - umožňuje žádat o jejich přidělení až na základě skutečné potřeby, a po jejich použití je zase vrátit. Konkrétní myšlenka je taková, že po celou dobu komunikace mezi klientem a serverem (tedy po celou dobu existence relace mezi nimi) musí existovat jen to, co zajišťuje vysílání a příjem nejrůznějších příkazů a odpovědí na ně mezi klientem a serverem, zatímco všechno ostatní, co je potřeba pro vlastní přenos dat, může vznikat dynamicky, až na základě skutečné potřeby.

Této představě odpovídá i vnitřní členění složek v roli klienta a serveru i povaha komunikačních kanálů mezi nimi - viz obr. 72.1.

Interpret protokolu a přenosový proces

Ta část složky v roli klienta, která existuje trvale (resp. po celou dobu práce s aplikací, která protokol FTP implementuje), je tzv. **interpret protokolu** (**PI, Protocol Interpreter**). Jeho úkolem je nejprve navázat spojení s partnerským interpretem protokolu na straně serveru, a poté iniciovat jednotlivé akce a řídit jejich průběh. Činí tak prostřednictvím příkazů, které zasílá svému partnerskému interpretu protokolu na straně serveru, a dále tím, že na své straně dynamicky vytváří a řídí tzv. **přenosový proces** (**DTP, Data Transfer Process**), který pak zajišťuje vlastní přenos dat na základě pokynů od svého interpretu protokolu a pokynů druhé strany.

.PI OBR72_1.TIF, 20, 40, 10

Obr. 72.1. Představa složek v roli klienta a serveru protokolu FTP

Řídící a datové spojení

Interpret protokolu na straně klienta navazuje spojení se svým partnerským interpretem na straně serveru prostřednictvím protokolu TCP - to znamená, že spojení mezi oběma interprety má povahu spojovaného a spolehlivého spojení. Interpret na straně serveru musí čekat na žádost o navázání spojení na jednom z tzv. dobře známých portů (konkrétně na portu č. 21). Jakmile je spojení navázáno, existuje po celou dobu existence relace, ale je využíváno jen pro pro potřeby řízení této relace - proto se mu také říká **řídící spojení** (**control connection**).

Jakmile si oba interprety mezi sebou vykorespondují, že má být proveden nějaký přenos, vytvoří za tímto účelem své přenosové procesy. Ty si pak mezi sebou naváží samostatné spojení (opět prostřednictvím protokolu TCP, tedy spolehlivé a spojované), a jeho prostřednictvím pak zajistí vlastní přenos dat. Proto se tomuto spojení také říká **datové spojení** (**data connection**). Za povšimnutí stojí skutečnost, že zatímco navázání řídícího spojení iniciuje klient (přesněji interpret protokolu na straně klienta), povinnost iniciovat navázání datového spojení má naopak server (resp. jeho přenosový proces). Interpret protokolu na straně klienta naopak instruuje svůj přenosový proces, aby čekal na zadaném portu na navázání spojení.

Protokol FTP ovšem připouští i jinou možnost, než jen přenos souborů mezi jedním klientem a jedním serverem. Dovoluje, aby uživatel z jednoho počítače inicioval přenos souborů mezi dvěma jinými počítači (tedy mezi dvěma vzdálenými počítači). V tomto případě existuje složka v roli klienta na počítači, na kterém pracuje uživatel, a na obou vzdálených počítačích musí existovat složky v roli serveru.Řídící spojení musí být navázána dvě, a to mezi interpretem protokolu klientské složky a interprety obou serverů, a jejich prostřednictvím instruuje klient oba servery, jak mají zajistit požadovaný přenos. Datové spojení je pak ale vytvářeno jen mezi oběma servery, tj. vlastní data již neprochází přes klientský počítač.

Příkazy pro řízení přenosu

Interprety protokolu na straně klienta a serveru spolu komunikují prostřednictvím jazyka, který je součástí definice protokolu FTP. Tento jazyk je tvořen příkazy (které vysílá interpret klienta) a odpověďmi na ně (které vysílá interpret serveru). Jednotlivé příkazy lze rozdělit do tří základních skupin, na:

- příkazy pro řízení přístupu (access control commands), umožňují mj. zadat uživatelské jméno a heslo, které pak složka v roli serveru bude používat při všech následných požadavcích na přístup k souborům (viz též 70. díl)

- příkazy pro nastavení parametrů přenosu (transfer parameter commands), které umožňují například změnit implicitní čísla portů, používaných při navazování datového spojení, nastavit režim přenosu, stanovit reprezentaci dat či strukturu přenášeného souboru (viz minulý díl seriálu).

- výkonné příkazy (FTP service commands), které iniciují vlastní přenosy souborů a manipulaci s nimi (přejmenovávání, rušení), práci s adresáři (přechody na podadresář či na nadřazený adresář), výpisy obsahu adresáře (vlastní výpis je pak přenášen prostřednictvím datového spojení) apod.

Odpovědi

Každý příkaz generuje alespoň jednu odpověď (některé dokonce více odpovědí). Jednotlivé odpovědi přitom mají formu číselných kódů - trojmístných desítkových čísel - za kterými následuje vysvětlující text. Představa je ovšem taková, že rozhodující jsou pouze číselné kódy, které v sobě nesou veškerou podstatnou informaci, zatímco text má pouze vysvětlující charakter, a jeho obsah není navíc vůbec závazný (tj. různé implementace mohou ke stejnému číselnému kódu připojovat různé vysvětlující texty). Logika je taková, že číselný kód je určen pro program, který odpovědi vyhodnocuje. Ten může podle svého uvážení vysvětlující text jednoduše ignorovat, nebo jej zobrazit uživateli, aby i on byl pro něj srozumitelnou formou informován o odpovědích serveru.

Zajímavý je i způsob kódování číselných odpovědí, který vychází vstříc různé úrovni detailnosti při vyhodnocování odpovědí. Číselný kód je trojmístný, a jeho první číslice udává to, zda odpověď je dobrá, špatná či neúplná (viz tabulka 72.2.). Nejjednodušší implementace se pak mohou rozhodovat jen podle této první číslice, zatímco propracovanější implementace mohou získat podrobnější informace vyhodnocením druhé, ev. i třetí číslice.

1xx předběžná kladná odpověď
(požadovaná akce byla úspěšně zahájena,
ještě bude následovat další zpráva o jejím průběhu)

2xx kladná odpověď
(požadovaná akce byla úspěšně provedena, resp.
dokončena)

3xx prozatímní kladná odpověď
(příkaz byl přijat, ale pro zahájení požadované akce jsou nutné ještě další příkazy)

4xx dočasná záporná odpověď
(požadovaná akce nebyla úspěšně provedena, ale důvod neúspěchu je dočasný, a má smysl žádat o nové provedení akce)

5xx trvalá záporná odpověď
(požadovaná akce nebyla úspěšně provedena a nemá smysl se o ni pokoušet znovu)

Tabulka 72.2: Význam první číslice odpovědi

FTP používá Telnet

Za povšimnutí stojí i konkrétní forma, v jaké jsou jednotlivé příkazy a odpovědi na ně přenášeny řídícím spojením. Příkazy i odpovědi mají zásadně textovou formu - jména příkazů tvoří tří až čtyřpísmenné zkratky, a také číselná část odpovědi je přenášena textově (jako trojice desítkových číslic). Vzhledem k tomu se protokol FTP může odvolávat na protokol Telnet a požadovat, aby tyto texty byly přenášeny po řídícím spojení přesně tak, jak to činí protokol Telnet. V praxi to znamená, že implementace protokolu FTP pro přenos příkazů a odpovědí řídícím spojením buďto sama využívá služeb již existující implementace Telnetu, nebo si sama implementuje

tu část protokolu Telnet, kterou skutečně potřebuje (protože zdaleka nevyužívá všech možností, na které protokol Telnet pamatuje).

Uživatelský vs. řídící jazyk

Důsledně textová podoba řídícího jazyka i využití protokolu Telnet pro přenos příkazů a odpovědí nevylučuje, aby na straně klienta vydával příkazy řídícího jazyka místo interpretu protokolu přímo uživatel, který si otevře vzdálenou terminálovou relaci s interpretem na straně serveru (pokud přitom dodrží všechny konvence protokolu FTP). Běžný případ to ale rozhodně není.

Obvyklá implementace protokolu FTP obsahuje vedle interpretu příkazů a přenosového procesu ještě i uživatelské rozhraní, které zajišťuje komunikaci mezi uživatelem a interpretem příkazů (viz obr. 72.1.). Toto rozhraní pak může nabízet v podstatě jakýkoli "uživatelský" jazyk, který pak ve spolupráci s interpretem protokolu překládá do řídícího jazyka, a ten již je pro všechny implementace jednotný. Z pohledu uživatele (který využívá služeb protokolu FTP prostřednictvím uživatelského jazyka) se proto různé implementace FTP mohou "tvářit" jinak (mohou používat různé příkazy pro tytéž činnosti, místo řádkového rozhraní mohou používat grafické uživatelské rozhraní apod.), ale díky používání jednotnéhořídícího jazyka by se měly všechny správně domluvit.

Příklad průběhu relace mezi klientem a serverem ukazuje obrázek 72.3.

.PI OBR72_3.TIF, 20, 40, 10

Obr. 72.3. Příklad průběhu relace protokolu FTP

73/ TFTP, anonymní FTP servery

V minulých dvou dílech jsme se podrobněji zabývali protokolem FTP pro netransparentní přenos v sítích na bázi TCP/IP. Již na začátku jsme si ale avizovali, že vedle tohoto "plnohodnotného" protokolu existuje ještě jeden, o mnoho jednodušší a skromnější, označovaný příznačně jako TFTP, neboli "Triviální FTP".

Jaká je ale logika, která stojí za existencí dvou protokolů pro netransparentní přenos souborů? Proč je výhodné mít dva?

Protokol FTP lze označit za plnohodnotný v tom smyslu, že je povinně vybaven mnoha různými funkcemi, které však nemusí být zdaleka vždy využívány. Množství různých funkcí a velký rozsah schopností však znamená větší, a obvykle i pomalejší program, který také zabírá více místa jak na disku, tak i při svém běhu v operační paměti. V dnešní době, kdy se objemy operačních pamětí počítají spíše na megabyty, by nějaký ten kilobyte nemusel hrát příliš velkou roli. Přesto jsou ale situace, kdy hraje roli významnou. Tak významnou, že se kvůli ní vyplatí použít jiný protokol, jehož implementace je méně náročná na různé systémové zdroje.

Co potřebují bezdiskové stanice

Pro příklad je nejlépe zajít k bezdiskovým pracovním stanicím. Nemaje vlastní pevný disk, jsou odkázány na sdílení disků jiných uzlových počítačů prostřednictvím sítě. Nejprve si ale musí odněkud načíst svůj vlastní operační systém a další "náležitosti",

aby se vůbec rozeběhly - z některého z uzlových počítačů, který plní roli tzv. bootovacího serveru, a svým klientům nabízí jako svou službu možnost nahrát si to, co potřebují ke svému startu a co by jinak měly na svém vlastním disku (kdyby jej ovšem měly). Aby však mohla bezdisková stanice (jako klient) tuto službu využít, musí umět se na příslušný server vůbec obrátit a komunikovat s ním. A to okamžitě po svém zapnutí, ještě dříve než od bootovacího serveru cokoli získá. Jediné možné řešení je pak umístit vše potřebné pro navázání komunikace se serverem do pevné paměti, tak aby příslušný kód mohl být spuštěn ihned po zapnutí stanice.

To, co pracovní stanice od bootovacího serveru získává, má formu souborů, a tak v této pevné paměti musí být implementován vhodný protokol pro přenos souborů, a s ním i všechny protokoly nižších vrstev, které ke své práci potřebuje. Je ovšem zcela zbytečné, aby to byl "plnohodnotný" protokol - jeho činnost je vždy stejná, a tak lze předem odhadnout, které funkce bude skutečně využívat a které nikoli, a implementovat jen ty skutečně potřebné. Velmi častý je i takový případ, kdy velmi malý kód v pevné paměti načte ze serveru a zavede do operační paměti stanice složitější zaváděcí program, a teprve ten pak zajistí načítání operačního systému a všeho potřebného pro start operačního systému na bezdiskové stanici (tomuto "vícestupňovému" zavádění se v angličtině říká **bootstrapping**).

Další motivací pro existenci jednoduchého protokolu pro přenos souborů pak jsou takové aplikace, které "vymoženosti" plnohodnotného protokolu nepotřebují, dále jednoduché operační systémy, kterým činí problémy udržovat současně více otevřených transportních spojení (jak to vyžaduje protokol FTP) apod.

V čem se liší TFTP

Protokol TFTP (Trivial File Transfer Protocol) se od protokolu FTP liší především v následujících aspektech:

- pro transport dat používá služeb protokolu UDP (zatímco FTP používá transportní služby protokolu TCP)

- nezajišťuje žádné systémové akce na vzdáleném počítači (typu výpisu adresáře, změny aktuálního adresáře, rušení souborů apod.),

- nezajišťuje žádnou identifikaci uživatele, který žádá o přenos

- přenáší jen textové a binární soubory.

Soubory, které přenáší, chápe protokol TFTP buď jako textové soubory, nebo jako soubory binární. V prvním případě předpokládá (obdobně jako protokol FTP), že jednotlivé znaky jsou při přenosu kódovány přesně takovým způsobem, jaký požaduje protokol Telnet (a příjemce či odesilatel je v případě potřeby konvertuje z/do místních konvencí), zatímco v druhém případě se na obsah přenášeného souboru dívá jako na posloupnost osmibitových bytů, a nijak je neinterpretuje.

Pro vlastní přenos využívá protokol TFTP nespolehlivých a nespojovaných služeb transportního protokolu UDP. S jeho nespolehlivostí se vyrovnává tak, že si sám zajišťuje potřebné potvrzování. Toto potvrzování je zajímavé tím, že jde o tzv. jednotlivé potvrzování (tedy takové, kdy odesilatel po odeslání jednoho bloku čeká na jeho explicitní potvrzení, a teprve pak vysílá další blok), a dále tím, že používá symetrické čekání na vypršení časového limitu (tzv. timeout). Odesilatel, který odeslal nějaká data, čeká na jejich potvrzení, a pokud jej nedostane do určitého časového

limitu, vyšle data znovu. Obdobně příjemce, který data obdrží, je potvrdí odesláním příslušného potvrzení, ale pokud do časového limitu nedostane další data, znovu vyšle již jednou odeslané potvrzení. Jednotlivé bloky dat, které jsou tímto způsobem přenášeny, mají pevnou velikost (512 bytů), a jsou sekvenčně číslovány od 1. Podobně jsou sekvenčně číslována i potvrzení. Konec celého přenosu příjemce rozpoznává podle posledního bloku, který musí obsahovat méně než standardních 512 bytů (tedy např. i 0 bytů).

V prvním (resp. nultém) bloku, kterým se přenos zahajuje, musí být uvedeno přesné jméno souboru, který má být přenesen, včetně úplné přístupové cesty k tomuto souboru. Protokol TFTP, na rozdíl od protokolu FTP, totiž nepočítá s tím, že by na straně serveru byl nastaven na nějaký konkrétní adresář. V důsledku toho ani nezná pojem aktuálního adresáře (na serveru), neumožňuje přecházet mezi jednotlivými adresáři, a nezprostředkovává ani jejich výpis. Počítá s tím, že klient přesně ví, kde a s jakým souborem chce pracovat.

Jménem koho?

Velmi zajímavé důsledky vyplývají z dalšího zjednodušení protokolu TFTP. Na rozdíl od "plnohodnotného" protokolu FTP totiž nezná pojem uživatele. Nepočítá s tím, že by uživatel v roli klienta prokazoval serveru svou totožnost (resp. toto neumožňuje), a tak všechny jeho požadavky na čtení či zápis jednotlivých souborů jsou vlastně anonymní. Jak má ale server posuzovat jejich oprávněnost či neoprávněnost?

Definice protokolu TFTP toto ponechává na implementaci, ale obvyklé řešení je takové, že číst lze jen ty soubory, které jsou ke čtení přístupné všem uživatelům. V případě zápisu pak rozhodují přístupová práva do konkrétního adresáře (zda umožňuje zápis všem uživatelům či nikoli), resp. přístupová práva ke konkrétnímu souboru (jde-li o přepis nebo o přidávání za konec již existujícího souboru).

Anonymní FTP servery

Na problém s přístupovými právy narazíme i v okamžiku, kdy se rozhodneme vytvořit na nějakém počítači veřejně přístupný archiv, ze kterého by si kdokoli mohl nahrávat zde umístěné soubory. Jaký protokol pro přenos souborů je k tomuto účelu nejvhodnější?

Protokol TFTP by byl vhodný právě pro svou "anonymitu", díky které by zájemci o přístup k archivu nemuseli uvádět žádná uživatelská jména ani hesla. Má to ovšem jeden malý háček - jelikož protokol TFTP neumožňuje procházet adresáři serveru a vypisovat si jejich obsah, nemohl by si zájemce sám vyhledávat to, co jej zajímá či co potřebuje, ale musel by být s obsahem archivu seznámen jiným způsobem, a pak jít "na jistotu".

Proto se ke zpřístupnění nejrůznějších archivů souborů v sítích na bázi TCP/IP používá "plnohodnotný" protokol FTP. I zde je ale malý háček - má-li být archiv opravdu veřejně přístupný, jak sdělit všem potenciálním zájemcům potřebné uživatelské jméno (a případně i heslo), které mají použít? Možným řešením je zvolit jedno konkrétní jméno, a to pak používat všude. Tedy vlastně zavést jednotnou konvenci, a tu důsledně dodržovat. No a jaká že tato konvence je? V síti Internet, která je dnes zdaleka nejvýznamnější sítí na bázi protokolů TCP/IP a pokrývá

prakticky celý svět, je touto jednotnou konvencí uživatelské jméno **anonymous** (doslova: anonym). Podle něj jsou pak také příslušné veřejně přístupné archivy označovány jako **anonymní FTP servery** (**anonymous FTP servers**). V celém Internetu jich je opravdu mnoho, jen jejich stručné seznamy mívají desítky stránek.

Pokud se na některý z nich obrátíte (v roli klienta protokolu FTP), můžete se přihlásit jako uživatel "anonymous". Heslo pak na vás již vůbec není požadováno, nebo jste vyzváni k tomu, aby jste místo hesla uvedli svou skutečnou identitu (míněno: svou adresu pro elektronickou poštu) - viz též obrázek 72.3 v minulém dílu seriálu. Některé anonymní FTP servery používají takto získanou informaci jen pro vlastní evidenci (aby měly přehled, kdo a jak je využívá). Jiné anonymní servery však mohou být méně důvěřivé, a ověřují si, zda vámi uvedená identita je skutečná, či nikoli. A pokud dojdou k závěru, že je smyšlená (nebo nejsou schopny její pravost ověřit), jednoduše se s vámi přestanou bavit.

74. Transparentní sdílení souborů

V posledních třech dílech jsme se podrobněji zabývali dvěma protokoly (FTP a TFTP). Ty mají v rodině protokolů TCP/IP na starosti takový přenos souborů, který jsme si označili jako netransparentní. Nyní je na řadě transparentní přístup k souborům a protokol NFS.

Nejprve si ale znovu zopakujme, v čem je rozdíl mezi transparentním a netransparentním přístupem (tak, jak jsme si je zavedli v 70. dílu): netransparentní přístup je takový, při kterém pro uživatele existuje principiální rozdíl mezi místními soubory a vzdálenými soubory. Uživatel si musí být vědom, že určitý soubor není místní, ale že se nachází na vzdáleném počítači (navíc musí vědět kterém), a když s ním chce pracovat, musí pro to nejprve něco udělat - zajistit přenos tohoto souboru ze vzdáleného počítače na svůj počítač.

Naproti tomu v případě transparentního přístupu pro uživatele neexistuje žádný principiální rozdíl mezi místními a vzdálenými soubory. Uživatel (resp. jím provozovaná aplikace) si může myslet, že všechny jemu dostupné soubory jsou místní, a jako s takovými s nimi také pracuje. Skutečnost, že některé soubory místní nejsou, je před uživatelem skryta, stejně tak jako všechny mechanismy, které potřebnou iluzi vytváří, a které zajišťují potřebné přenosy celých souborů či jejich částí mezi místním a vzdáleným počítačem.

Jak vzniká iluze

Principiální způsob, jakým lze transparentní přístup ke vzdáleným souborům realizovat, ukazuje obrázek 74.1.: vychází opět z modelu klient/server, ve kterém uživatelův počítač vystupuje v postavení klienta, a vzdálený počítač v postavení serveru (file serveru, který jako službu nabízí svým klientům uchovávání souborů). Pro uživatele (resp. jím provozovanou aplikaci) je ovšem i tento vztah transparentní - uživatel vznáší veškeré své požadavky na přístup k souborům jedné a téže entitě (složce operačního systému, kterou prozatím nazveme **shell**). Ta rozhoduje o tom, zda požadavek je požadavkem na přístup k takovému souboru, který je lokální, nebo zda jde o požadavek na přístup ke vzdálenému souboru, a podle toho pak příslušný požadavek předá k vyřízení buď místnímu systému souborů, nebo programové entitě, která implementuje klienta. Pokud nastal tento druhý případ, klient zformuluje požadavek na přístup ke vzdálenému souboru do takového tvaru, který je

srozumitelný pro entitu v roli serveru na vzdáleném počítači, a požadavek jí odešle. Server zajistí vše potřebné pro vyřízení požadavku (např. načtení souboru, který pro něj již je místní), a výsledek odešle po síti zpět klientovi. Ten jej pak vrátí zpět shellu, a ten zase uživateli.

Praktické naplnění tohoto schématu má dvě relativně samostatné stránky: tou první je konkrétní protokol, prostřednictvím kterého komunikuje klient se serverem, a tou druhou je plně transparentní "začlenění" mechanismu přístupu ke vzdáleným souborům do celkového systémového prostředí na klientském počítači. Právě u této druhé stránky se nyní zastavíme podrobněji, zatímco tu první si ponecháme na další díly.

.PI OBR74_1.TIF, 20, 40, 10

Obr. 74.1.: Představa přístupu ke vzdáleným souborům

Záleží na operačním systému

Máme-li na určitém počítači implementovat takový mechanismus přístupu ke vzdáleným souborům, který má být pro uživatele plně transparentní, pak se nutně musíme přizpůsobit tomu, jaké konvence, postupy a mechanismy používá pro přístup k souborům operační systém daného počítače. Má-li totiž uživatel pracovat i se vzdálenými soubory přesně stejným způsobem, jako se soubory místními, musí k tomu využívat přesně stejné prostředky a postupy.

Jestliže tedy konkrétní protokol pro vlastní přenos souborů mezi uzlovými počítači může (či spíše musí) být stejný, způsob jeho zpřístupnění uživatelům a jimi provozovaným aplikacím je již závislý na konkrétním počítači, přesněji na jeho operačním systému.

Podívejme se proto, jak vypadá situace v případě dvou operačních systémů - Unixu a MS DOSu.

Unix montuje

Operační systém Unix si organizuje všechny své soubory do jediného adresářového stromu. Fyzicky mohou být tyto soubory umístěny na různých zařízeních (pevném disku, disketě apod.), a sdruženy do tzv. souborových systémů (filesystems) - například obsah diskety může tvořit takovýto souborový systém, samostatný oddíl (partition) pevného disku může tvořit jiný souborový systém apod. Z pohledu uživatele jsou ale tyto souborové systémy sestavovány do jednoho jediného adresářového stromu, jehož základ tvoří tzv. kořenový souborový systém (root filesystem). Konkrétní mechanismus, který toto sestavování umožňuje, je operace **mount** (doslova: připoj, přimontuj). Z pohledu uživatele funguje tak, že celý souborový systém připojí k zadanému adresáři globálního adresářového stromu jako jeho nový podstrom - viz obrázek 74.2.

.PI OBR74_2.TIF, 20, 40, 10

Obr. 74.2.: Představa operace mount v OS Unix

Když se pak uživatel či jakákoli aplikace odkazuje na některý soubor z takto "přimontovaného" systému souborů, lze si představit, že jeho požadavek projde od kořene až do místa, kde je příslušný systém souborů připojen, a odsud je pak předán ovladači připojeného systému souborů, který tento požadavek již dokáže vyřídit - viz obr. 74.3.

Takto fungující prostředí vychází vstříc plně transparentnímu přístupu ke vzdáleným souborům, a umožňuje realizovat jej velmi přirozeným způsobem. Stačí rozšířit schopnosti operace mount tak, aby umožňovala připojit i takové systémy souborů či jejich části, které se fyzicky nachází na vzdáleném počítači, a dále zajistit, aby veškeré požadavky na přístup k těmto souborům se na daném počítači dostaly k entitě, která implementuje klienta - viz obr. 74.3. Ta pak již zajistí vše potřebné.

.PI OBR74_3.TIF, 20, 40, 10

Obr. 74.3.: Přístup k místním a vzdáleným souborům v Unixu

DOS trvá na logických jednotkách

Jestliže operační systém Unix sestavuje všechny systémy souborů vždy jen do jediného stromu, který pak nepotřebuje nijak pojmenovávat, operační systém MS DOS se na své adresáře a v nich obsažené soubory dívá poněkud jiným pohledem. To, co Unix sestavuje pomocí operace mount do jediného celku, ponechává MS DOS oddělené - obsah diskety v každé z disketových mechanik chápe jako samostatný celek, a stejně tak každý oddíl (partition) pevného disku či tzv. RAMdisk. Každý takovýto celek (v terminologii DOS-u nazývaný **drive**, a představující logickou diskovou jednotku) musí být jednoznačně identifikovatelný, a tak mu DOS přiřazuje jednopísmenné označení: například drive A: je první disketová jednotka, B: případná druhá disketová jednotka, C: první oddíl (partition) prvního pevného disku. Navíc nemá MS DOS žádnou obdobu Unixovské operace mount (a ani ji nepotřebuje).

V prostředí MS DOSu proto musí být plně transparentní přístup ke vzdáleným souborům implementován tak, aby systémy souborů vzdáleného počítače či jejich podstromy vystupovaly jako samostatné celky (logické jednotky, drives) - viz obr. 74.4.

Dalším nedostatkem MS DOSu je, že nedokáže vždy potřebným způsobem rozeznávat požadavky na přístup ke vzdáleným souborů a předávat je tomu, kdo je dokáže vyřídit. Proto je obvykle třeba tyto požadavky přesměrovávat ještě dříve, než se dostanou k MS DOSu - vrstvou, která "překryje" DOS, zachytává všechny požadavky na přístup k souborům, ty "místní" předává DOSu, a pro ostatní pak ve spolupráci s dalšími entitami (implementujícími klienta) zařizuje vše potřebné. No a právě touto překrývající vrstvou je entita, kterou jsme si dříve označili jako shell (což v doslovném překladu znamená plášť). V prostředí MS DOSu, který nepodporuje multitasking, musí mít tato entita formu rezidentního programu.

.PI OBR74_4.TIF, 20, 40, 10

Obr. 74.4.: Pohled na vzdálené soubory v prostředí MS DOSu

75/ NFS I.

V minulém dílu jsme se začali podrobněji zabývat problematikou plně transparentního přístupu ke vzdáleným souborům v počítačových sítích. Na příkladu operačních systémů Unix a MS DOS jsme si naznačili jeden z aspektů této problematiky - principiální způsob "navázání" mechanismů, které přístup ke vzdáleným souborům zajišťují, na konkrétní operační systém. Dnes se již začneme podrobněji věnovat jednomu konkrétnímu mechanismu, protokolu NFS.

Protokol NFS (Network File System) má v síťovém modelu a rodině protokolů TCP/IP poněkud specifické postavení. Jak jsme si uvedli již ve 42. dílu, "klasické" protokoly rodiny TCP/IP (např. IP, TCP, UDP apod.) pochází převážně z akademického prostředí. Vytvořily je kolektivy odborníků z univerzit USA v rámci programu, který financovala grantová agentura DARPA ministerstva obrany USA, a který vyústil ve vznik dnes již celosvětové sítě Internet, založené právě na těchto protokolech. Díky způsobu svého vzniku mohou být protokoly TCP/IP tzv. veřejným vlastnictvím (public domain), neboť v USA praktikují velmi spravedlivou zásadu: co bylo vytvořeno za peníze daňových poplatníků, to je také jejich vlastnictvím. V praxi to znamená, že tyto standardy "nepatří" žádné konkrétní instituci či dokonce komerční firmě, která by přístup k nim mohla komukoli omezovat, či dokonce požadovat za jejich zpřístupnění nějaké licenční poplatky. Jsou naopak běžně dostupné pro kohokoli, kdo o to projeví zájem, a to buď úplně zdarma, nebo jen za nezbytný manipulační poplatek.

Existuje pouze organizace, zajišťující administrativní agendu kolem zveřejňování těchto protokolů - je jí tzv. *Network Information Center* (zkratkou: NIC), financovaná opět z prostředků ministerstva obrany USA. Konkrétní formou kodifikace protokolů TCP/IP (stejně tak jako dalších mechanismů, zásad či principů, používaných v síti Internet) jsou dokumenty, označované jako **RFC** (Request For Comment, doslova: žádost o poznámky, resp. připomínky). Tyto dokumenty jsou sekvenčně číslovány, a jsou volně dostupné například prostřednictvím sítě Internet (ve formě textových souborů).

NFS je od Sun-ů

Protokol NFS se od tohoto schématu odlišuje tím, že vznikl jako vlastnířešení jedné komerční firmy - firmy **Sun Microsystems, Inc.**, známého výrobce Unixovských pracovních stanic. Z tohoto pohledu by tedy bylo nutné označit protokol NFS za proprietární (proprietary).

Firma Sun Microsystems však koncipovala svůj protokol velmi důsledně jako otevřený - nejen po stránce technické (jako protokol, který lze implementovat pod různými operačními systémy a na různých počítačích), ale i po stránce autorsko-právní: zcela záměrně zveřejnila přesné specifikace protokolu NFS (i dalších dvou podpůrných mechanismů, RPC a XDR, ke kterým se ještě dostaneme), tak aby je mohli využívat i jiní výrobci, a vytvářet vzájemně kompatibilní systémy. Díky nesporným kvalitám samotného protokolu a díky prozíravé strategii firmy Sun se její očekávání naplnilo, a protokol NFS se stal velmi oblíbeným mechanismem pro

implementaci plně transparentního přístupu ke vzdáleným souborům v počítačových sítích.

V současné době, díky jeho velkému rozšíření v sítích na bázi protokolů TCP/IP, lze protokol NFS považovat za součást rodiny protokolů TCP/IP. Svědčí o tom i skutečnost, že specifikace tohoto protokolu jsou zveřejněny formou dokumentu RFC (konkrétně RFC 1094). Příznačná je i skutečnost, že o tomto protokolu se nejčastěji hovoří jako o "protokolu, který vyvinula firma Sun", a nikoli jako o "protokolu firmy Sun".

Prozíravá autorskoprávní politika firmy Sun jistě velmi přispěla k výraznému úspěchu protokolu NFS, ale sama o sobě by zdaleka nestačila k jeho prosazení do života. O to se musely přičinit především přednosti protokolu jako takového. A jaké že tyto přednosti jsou?

NFS vs. Unix

Protokol NFS rozhodně nezapře, že byl vytvořen v Unixovském prostředí. Firma Sun jej ale nekoncipovala jako síťové rozšíření svého operačního systému SunOS (vlastní verze Unixu, vycházející z větve tzv. BSD Unixu), protože to by vylučovalo jeho využití jinými výrobci a možnost spolupráce s jejich produkty. Místo toho jej od začátku pojala jako univerzální síťové rozšíření, které není vázáno na určitý konkrétní operační systém. Protokol NFS tedy může být (a v praxi také skutečně je) implementován snad pro všechny "příchuti" Unixu. Není však zdaleka omezen je na operační systémy Unixovského typu. Je natolik univerzální, že může být implementován i pod jinými operačními systémy, MS DOS nevyjímaje (i když právě zde se, vzhledem k jednouživatelské a jednoúlohové povaze DOSu, lze setkat jen s implementacemi klientů, a nikoli serverů). Díky tomu pak může být protokol NFS úspěšně využíván jako prostředek pro plně transparentní sdílení souborů nejen mezi počítači s různými variantami Unixu, ale také mezi počítači, které stojí na zcela odlišných platformách - například Unixovský počítač může sloužit jako file server v lokálních sítích s počítači PC, provozujícími MS DOS.

Bezestavová filosofie

Jedním z klíčů k výraznému úspěchu protokolu NFS jistě bylo to, že je důsledně koncipován jako bezestavový (anglicky: stateless). Co to znamená a k čemu to je dobré?

Představme si nejprve opačný případ - tedy stavový protokol. Ten předpokládá, že klient i server se mohou nacházet v různých stavech, a v závislosti na průběhu své komunikace mezi nimi různě přecházet. Například když si klient vyžádá otevření souboru XY, server jeho žádosti vyhoví a soubor otevře. Tím ovšem přejde z jednoho stavu (který by bylo možné charakterizovat jako: soubor XY není otevřen) do jiného stavu (stavu: soubor XY je otevřen). Když si pak klient vyžádá uzavření souboru XY, server opět změní svůj stav atd.

S existencí různých stavů je ovšem spojena určitá stavová informace - ve výše uvedeném příkladu si server musí pamatovat, zda si příslušný klient otevřel soubor XY (a kromě toho i jakým způsobem atd.), nebo nikoli. A právě uchovávání této stavové informace může být velkým problémem. Může se totiž stát, že buď klient, nebo server náhle o tuto stavovou informaci přijdou - ať již vlivem výpadku spojení či

v důsledku "pádu" počítače a jeho operačního systému (např. kvůli náhlému výpadku proudu, tzv. přebootování či z jiné příčiny).

Pokud ovšem jedna ze zúčastněných stran náhle přijde o svou stavovou informaci, neví, v jaké fázi komunikace s protistranou se právě nachází, a proto v této komunikaci nemůže dost dobře pokračovat. Existují samozřejmě způsoby, jak zajistit potřebnou rekonvalescenci (uzel po výpadku může například rozeslat hlášení typu: "všechno je špatně, začínáme znovu"), ale příslušné mechanismy nejsou zdaleka bezproblémové (co když se například příslušné hlášení ztratí?), a navíc jsou dosti neefektivní.

Celá problematika zajištění korektní stavové komunikace mezi serverem a klientem je sice řešitelná, ale je značně netriviální. Firma Sun se s touto netriviálností vyrovnala tak, že stavový charakter komunikace vyloučila.

Protokol NFS je tedy bezestavový v tom smyslu, že server se po provedení jakéhokoli požadavku klienta nachází v přesně stejném stavu, jako před příchodem tohoto požadavku. V důsledku toho si pak nemusí pamatovat nic o průběhu či stavu komunikace s kterýmkoli klientem, a po případném výpadku či ztrátě spojení nemusí být prováděny žádné nápravné akce. Pokud nějaký klient přijde se svým požadavkem v době, kdy server je mimo provoz (či "spadne" právě v době, kdy takovýto požadavek zpracovává), klientovi stačí jeho požadavek opakovat až doby, než server znovu "naběhne". Pokud se klient do výpadku serveru "nestrefí", nemusí si tento výpadek vůbec uvědomit.

Šanci mají jen idempotentní

Bezestavovost tedy sebou přináší velkou robustnost (tj. odolnost proti negativním vlivům, v tomto případě výpadkům). Není to ovšem zadarmo. Cenou, kterou se za tuto robustnost platí, je možnost požadovat na serveru jen takové operace, které se v matematice označují velmi přesným, ale zato nepříliš libě znějícím přívlastkem: **idempotentní**. Co se tím míní?

Idempotentní je taková operace, kterou lze opakovat s přesně stejným efektem, jaký mělo její první provedení. Ukažme si to na příkladu: operace "načti z daného souboru X bytů, počínaje pozicí Y" je idempotentní, protože při každém svém provedení dá vždy stejný výsledek (a můžeme ji tedy vícekrát opakovat). Naproti tomu operace "načti z daného souboru dalších X bytů" idempotentní není, protože při každém jejím provedení budou načteny jiná data (dokud se nenarazí na konec souboru). Tuto druhou operaci může klient požadovat jen na stavovém serveru, který si jako svou stavovou informaci musí pamatovat, kam až příslušný klient "dočetl" daný soubor.

Jaký charakter má práce se soubory?

Jestliže jsou možné požadavky klienta na server omezeny jen na idempotentní operace, pak se jistě nabízí zajímavá otázka: je možné všechny potřebné akce se soubory realizovat pomocí idempotentních operací? Odpověď na tuto otázku je bohužel záporná.

Například obvyklé schéma práce se soubory, stylem "otevři soubor, zapisuj resp. čti, zavři soubor", je svou podstatou stavové (kvůli možným stavům dotyčného souboru). Lze jej ovšem převést na bezestavové - tím, že odstraní explicitní otevírání a zavírání souborů, a nahradí se implicitním otevřením a následným uzavřením v rámci každého

jednotlivého čtení či zápisu. Proto také protokol NFS nenabízí žádné prostředky pro otevírání či zavírání souborů.

Existují ovšem i takové druhy akcí, které při nejlepší vůli není možné převést na idempotentní - příkladem může být tzv. APPEND (neboli připojování dat za aktuální konec souboru), nebo uzamykání souborů či jejich částí pro potřeby vícenásobného přístupu (například když jeden klient potřebuje, aby mezi dvěma jeho přístupy nemohl k témuž souboru přistupovat někdo jiný). Tvůrci protokolu NFS se s tímto problémem mohli vyrovnat v podstatě jediným možným způsobem - zákazem toho, bez čeho se lze obejít (což je příklad tzv. Append-u), či ponecháním takovýchto funkcí "vně" protokolu NFS (tedy tím, že je svěřili samostatným mechanismům, resp. protokolům, jako například uzamykání souborů).

Bezestavový může být efektivní

Další velkou výhodou bezestavového protokolu jsou menší požadavky na spolehlivost přenosových služeb, než jaké nutně musí mít protokol stavový. Ten totiž musí zajistit, aby všechny požadované operace byly prováděny vždy právě jednou, a navíc ještě ve správném pořadí. K tomu ovšem nutně potřebuje spolehlivé přenosové služby spojovaného charakteru. Naproti tomu bezestavový protokol nemusí klást žádné specifické požadavky na kvalitu a charakter transportních služeb, a může využít i nespolehlivé, ale zato rychlé přenosové protokoly.

I to je jeden z důvodů, který napomohl značné popularitě bezestavového protokolu NFS - jelikož nemá žádné specifické požadavky, dokáže vystačit prakticky s kterýmkoli transportním protokolem, který je k dispozici, a může si vybrat ten, který je nejrychlejší. V prostředí TCP/IP sítí obvykle využívá přenosových služeb protokolu UDP.

76/ NFS II.

V minulém dílu jsme si začali podrobněji popisovat myšlenky a principy, na kterých je založen protokol NFS (Network File System), vyvinutý firmou Sun Microsystems. Nejprve jsme se věnovali bezestavovému charakteru celého protokolu a tomu, co z této jeho bezestavovosti vyplývá. Dnes se zaměříme na další zajímavé aspekty, které nám umožní pochopit, jak tento velmi populární protokol funguje "uvnitř".

Nejprve si ale znovu připomeňme další významnou charakteristiku protokolu NFS, která jeho tvůrcům ležela velmi na srdci - a to možnost implementovat tento protokol na různých platformách, v prostředí nejrůznějších operačních systémů. Bezestavový charakter protokolu NFS tomuto požadavku vychází vstříc mj. tím, že zcela záměrně klade minimální nároky na schopnosti a spolehlivost přenosových protokolů, a dokáže vystačit prakticky s jakýmkoli přenosovým protokolem, který je k dispozici.

Požadavek na snadnou implementaci v různých prostředích, resp. na různých platformách, se ovšem netýká jen klientů, ale samozřejmě také serverů. Dnes je vcelku běžné, že v roli file serverů systému NFS vystupují také různé ne-Unixovské počítače. Například střediskové počítače IBM s operačním systémem VM mohou být vybaveny protokoly TCP/IP včetně NFS, a stejně tak může v roli NFS serveru vystupovat například i server systému Novell NetWare - k tomu stačí zakoupit a nainstalovat na NetWare serveru doplňkový program, tzv. NetWare NFS. Ten sice

není zdaleka zadarmo, ale pokud lze věřit reklamním sloganům firmy Novell, vyjde toto řešení laciněji, než tradiční Unixovský počítač v roli NFS serveru.

Jestliže tedy protokol NFS může být implementován v prostředí různých operačních systémů, pak se nutně musí nějakým způsobem vyrovnat s jejich odlišnostmi. O tom, jaká může být povaha těchto odlišností, jsme si již něco naznačili v 70. dílu - na příkladu odlišností mezi Unixem a MS DOSem. Tvůrci protokolu NFS se se všemi potenciálními odlišnostmi rozhodli vyrovnat zajímavým způsobem: koncipovali samotný protokol NFS tak, aby se jej tyto odlišnosti vůbec netýkaly (a ošetření těchto odlišností svěřili jiným protokolům, které stojí mimo NFS).

Pouze přes systémovou identifikaci

Jedním z důsledků tohoto přístupu je skutečnost, že samotný protokol NFS nikdy nepracuje s přístupovými cestami. NFS serveru nemůže jeho klient zadat požadavek typu "načti X bytů ze souboru /usr/pet/file1.doc, počínaje pozicí Y", který určuje požadovaný soubor včetně přístupové cesty k němu. Důvodem je skutečnost, že různé operační systémy mohou používat různé konvence pro sestavování i pro formální zápis těchto přístupových cest. Obdobně je tomu i se jmény (a příponami) souborů - také zde používají různé operační systémy různé konvence.

Bylo by samozřejmě možné zavést nějakou jednotnou konvenci (například tu, kterou používá Unix), a pak zajistit případný převod z/do konvence, kterou používá konkrétní hostitelský operační systém. Tvůrci protokolu NFS se však rozhodli pro jiné řešení: pokud klient požaduje na serveru nějakou akci s určitým souborem, pak mu tento konkrétní soubor neurčuje jeho jménem, případnou příponou a přístupovou cestou, ale pomocí jednoznačného identifikátoru, který se označuje jako **file handle** (česky nejspíše: **systémová identifikace**), a je zcela nezávislý na jakékoli místní konvenci pro pojmenovávání souborů a sestavování přístupových cest.

Pro klienta je systémová identifikace posloupností bitů, které nijak neinterpretuje - jejich význam je pro něj irrelevantní (klient tedy může považovat systémovou identifikaci například za celé číslo bez znaménka). Z pohledu serveru se však systémová identifikace (file handle) skládá ze tří částí, které po řadě identifikují systém souborů, konkrétní soubor a jeho instanci (viz obr. 76.1.).

.PI OBR76_1.TIF, 20, 40, 10

Obr. 76.1: Představa systémové identifikace

.cp20

Konkrétní hodnotu jednotlivých bitů systémové identifikace samozřejmě určuje server, zatímco klient tuto identifikaci pouze používá, a chápe ji jako jediný, dále nedělitelný celek. Systémové identifikace, jednoznačně určující konkrétní soubory, pak ovšem musí klientovi poskytovat server, protože pouze on je dokáže vytvořit. Podívejme se nyní na formu, jakou se tak děje. Nejprve si ale řekněme, že systémová identifikace (file handle) může reprezentovat jak konkrétní soubor, tak i adresář.

Nyní si již představme situaci, kdy klient má k dispozici systémovou identifikaci nějakého konkrétního adresáře. Jednou z operací, kterou může na serveru požadovat,

je vypsání jmen všech souborů (a podadresářů) v tomto adresáři (tj. v tom, od kterého již klient vlastní systémovou identifikaci). Server mu vrátí seznam znakových řetězců a různých příznaků, ze kterých lze mj. poznat, zda znakový řetězec je jménem souboru či jménem podadresáře. Jestliže se pak klient rozhodne pracovat s nějakým konkrétním souborem v tomto adresáři, musí nejprve získat jeho systémovou identifikaci. To udělá prostřednictvím další možné operace, kterou si vyžádá na serveru, a které předá jako vstupní parametr jednak systémovou identifikaci adresáře, a dále řetězec se jménem souboru, který získal při předcházejícím výpisu obsahu tohoto adresáře. Stejně bude klient postupovat i v případě, kdy bude potřebovat systémovou identifikaci některého z podadresářů daného adresáře.

Jak tomu ale bude v případě, kdy klient potřebuje "projít" částí adresářového stromu, aby se dostal k nějakému konkrétnímu souboru? Zde je dobré si znovu explicitně zdůraznit, že vzhledem ke svému bezestavovému charakteru nemůže mít protokol NFS žádnou analogii "aktuálního adresáře", ve kterém by se klient mohl nacházet. Klient však může vlastnit systémové identifikace jednoho či několika adresářů, a má k dispozici mechanismy, pomocí kterých může získávat systémové identifikace i jejich podadresářů (či naopak nadřazených adresářů). Tímto způsobem se může postupně dopracovat až k systémové identifikaci adresáře, ve kterém se nachází požadovaný soubor, a získat také jeho systémovou identifikaci. Musí si ovšem sám postupně vybudovat "představu" příslušného adresářového stromu, protože server mu nabízí vždy jen pohled na jedno patro (resp. jednu úroveň) tohoto stromu.

Kde začít

Jakmile tedy klient vlastní systémovou identifikaci alespoň jednoho adresáře nějakého adresářového stromu, dokáže si jejím prostřednictvím získat systémové identifikace všech ostatních adresářů a souborů v daném adresářovém stromu, který mu jeho server zpřístupňuje. Důležité je přitom to, že server nikdy nepotřebuje sestavovat jména adresářů do celé přístupové cesty, a ani k nim připojovat jména souborů s jejich případnými příponami. Klientovi předává pouze jednorozměrné znakové řetězce, a ponechává na něm, zda a jak si je bude sestavovat do větších celků.

Zajímavou otázkou je ale to, jakým způsobem klient získá onu první systémovou identifikaci. Zde je dobré si uvědomit, že server nabízí svým klientům vždy celé adresářové stromy, které se v Unixové terminologii označují jako souborové systémy (file systems). Každý server vždy "zveřejňuje" seznam těchto souborových systémů (tzv. je exportuje), a každý klient si z nich může vybrat ty, které si pomocí operace **mount** připojí ke svému globálnímu adresářovému stromu (pokud je sám Unixovským počítačem), resp. které si zpřístupní jako samostatná logická zařízení (jde-li např. o klienta, který pracuje pod operačním systémem MS DOS) - jak jsme si podrobně popisovali v 74. dílu seriálu.

Konkrétní postup "připojování" je takový, že klient si nejprve vybere ze seznamu souborových systémů exportovaných určitým serverem ten, který jej zajímá, a pak tomuto serveru vyšle požadavek na jeho připojení (provedení operace **mount**). Položky seznamu přitom tvoří jména kořenů jednotlivých adresářových stromů (souborových systémů), včetně přístupových cest k nim. Požadavek na připojení (mount), který klient adresuje serveru, tudíž obsahuje jména včetně přístupových cest, a nutně tedy musí používat nějakou konvenci pro jejich sestavování. Požadavek

klienta ovšem není adresován přímo té entitě, která implementuje NFS server. Jejím adresátem je tzv. **mount server** (doslova: připojovací server), který není součástí NFS, a který již musí být uzpůsoben konkrétní konvenci pro pojmenovávání adresářů a sestavování přístupových cest, používané v prostředí, ve kterém je server implementován.

Tento mount server je tedy tím, kdo klientovi poskytne první systémovou identifikaci, se kterou se pak již klient může obracet na NFS server a od něj získávat další systémové identifikace - jak naznačuje i obrázek 76.2.

V tuto chvíli se ale nutně vnucuje otázka, proč je tomu právě takto? Jaká je logika v tom, že tzv. mount server není součástí NFS serveru (a způsob jeho činnosti není definován protokolem NFS)?

Důvody jsou především praktické. Služby mount serveru jsou využívány relativně velmi zřídka (typicky jednorázově při spuštění operačního systému klienta, který si v rámci svého startu připojuje souborové systémy serveru). Naopak služby NFS serveru mohou být využívány velmi často (při všech operacích se vzdálenými soubory). Na implementaci NFS serveru jsou proto kladeny velmi velké nároky, pokud jde o jeho rychlost, zatímco v případě mount serveru není jeho rychlost rozhodující. V prostředí Unixu je proto NFS server implementován jako součást jádra (aby mohl být co možná nejrychlejší), zatímco mount server může být implementován mimo jádro operačního systému, na stejné úrovni, jako běžné aplikační úlohy. Takovéto programy se ale píší a hlavně ladí mnohem snáze, než systémové programy, které mají být součástí jádra.

.PI OBR76_2.TIF, 20, 40, 10

Obr. 76.2. Systémové identifikace a přístupové cesty při komunikaci klienta se serverem

Mount server ovšem plní i některé další úkoly, než jen převádět jména adresářů včetně přístupových cest na systémové identifikace. Je to právě on, kdo skutečně "zveřejňuje" jména souborových systémů, exportovaných serverem. Má také za úkol ověřovat identitu uživatelů, kteří si ze svých počítačů v roli klientů připojují souborové systémy, a dále má i kontrolovat dodržování přístupových práv k exportovaným souborovým systémům. Díky tom, že si pro každého uživatele udržuje seznam jeho požadavků na připojení souborových systémů, je mount server nutně stavový (na rozdíl od bezestavového NFS serveru). Charakter stavové informace, kterou si mount server udržuje, však není kritický pro správné fungování NFS serveru i jeho klientů. Její význam je spíše pomocný, a využívá se například k tomu, aby server mohl včas varovat všechny své klienty o tom, že bylo zahájeno jeho řádné ukončení (tzv. shutdown).

77/ NFS - III.

V minulém dílu jsme se začali podrobněji zabývat tím, jak protokol NFS (Network File System) ve skutečnosti pracuje. Probrali jsme si princip komunikace klienta se serverem, pokud jde o způsob určování konkrétních souborů a adresářů - prostřednictvím systémových identifikací, a nikoli prostřednictvím jmen a přístupových cest k souborům. Dnes pokročíme ještě

dále a naznačíme si, jaké je postavení systémových identifikací (a protokolu NFS obecně) v rámci operačního systému počítačů v roli klientů i serverů.

Jelikož se naše dosavadní povídání o protokolu NFS týkalo především charakteru komunikace mezi klientem a serverem (neboli toho, co si vzájemně předávají, co od sebe požadují a co si poskytují), tedy jejich "vnějšími" projevy, nemuseli jsme náš výklad vztahovat k některé konkrétní systémové platformě, na které může být protokol NFS implementován. Jakmile se ale začneme podrobněji zabývat postavením NFS v rámci operačního systému, naše povídání již přestane být nezávislé na konkrétní platformě. Proto si musíme některou platformu vybrat, a vše potřebné si ukázat na jejím konkrétním příkladu (s naznačením odlišností pro jiné platformy). Nejvhodnějším platformou jistě bude prostředí, ve kterém protokol NFS vznikl, a ze kterého se pak rozšířil i na jiné platformy - tedy operační systém Unix.

Unix používá INODES

Jestliže protokol NFS má zajišťovat plně transparentní sdílení souborů (v tom smyslu, jaký jsme si zavedli již v 70. dílu), pak pro aplikační programy nesmí existovat žádný viditelný rozdíl mezi místními a vzdálenými soubory.

V prostředí operačního systému Unix pracují aplikační úlohy se soubory způsobem, který ukazuje obrázek 77.1. - prostřednictvím systémových volání (system calls), které směřují do jádra operačního systému (kernel), a přes něj pak do systému souborů (file system). To je tvořeno společnou částí, nezávislou na konkrétním druhu souborů, a dále různými ovladači, které již jsou implementačně závislé. Mezi těmito dvěma vrstvami je pak rozhraní, ve kterém se nachází datové struktury, popisující jednotlivé soubory - tzv. **INODES** (od: **Information Nodes**, někdy též: **Index Nodes**). V těchto datových strukturách jsou obsaženy například informace o přesném umístění souboru či adresáře, o jeho velikosti, vlastníkovi, o přístupových právech apod. Přesný význam i repertoár těchto informací je však "šit na míru" jednomu konkrétnímu způsobu uchovávání místních souborů, zatímco vzdálené soubory, sdílené prostřednictvím sítě, vyžadují přeci jen něco jiného. Odlišnosti zde sice nejsou nijak velké, ale na druhé straně zase nejsou zanedbatelné a stačí na to, aby datové struktury INODE nebyly příliš vhodné pro současné reprezentování jak místních, tak i vzdálených souborů.

.PI OBR77_1.TIF, 20, 40, 10

Obr. 77.1.: Představa rozhraní INODE (bez NFS)

Rozhraní VFS/VNODE

Firma Sun, která protokol NFS vyvinula, vyřešila celý problém následujícím způsobem: rozhraní, ve kterém se nachází datové struktury INODE, "obalila" ještě jednou vrstvou - nazývanou **Virtual File System (VFS) / Virtual File Node (VNODE) interface**, a původní datové struktury INODE "překryla" poněkud obecnějšími datovými strukturami, které již mohou reprezentovat jak vzdálené soubory, tak i soubory místní. Tím ještě důsledněji oddělila společnou část systému souborů (nezávislou na konkrétní implementaci) od její implementačně závislé části.

Výsledný efekt je takový, že nové rozhraní VFS/VNODE, vložené mezi obě tyto části, může být jednotné i v případě, kdy "překrývá" různě implementované systémy souborů - a to nejen vzdálené soubory, sdílené prostřednictvím protokolu NFS, ale také například místní soubory, uchovávané podle "zvyklostí" různých operačních systémů, v rámci kterých je protokol NFS implementován.

Jestliže tedy dříve pracovala společná část systému souborů přímo s rozhraním, obsahujícím struktury INODE (a označovaným jako "rozhraní INODE"), nyní již pracuje pouze s rozhraním VFS/VNODE a s datovými strukturami, které jsou v tomto rozhraní obsaženy. Původní struktury INODE sice nadále existují, ale systémová volání se k nim neobracejí přímo (ale pouze zprostředkovaně, přes datové struktury v rozhraní VFS/VNODE). V prostředí Unixu to samozřejmě znamenalo změnit všechna systémová volání, která zajišťují práci se soubory. Na druhé straně však toto řešení vyšlo vstříc nejen potřebám sdílení vzdálených souborů, ale také různým způsobům implementace souborových systémů v nejrůznějších "příchutích" Unixu (které mohou používat poněkud odlišné struktury INODE, a jednotné struktury v rozhraní VFS/VNODE).

Datové struktury VFS a VNODE

Datové struktury, obsažené v novém rozhraní, se jmenují příznačně: VFS a VNODE. Struktura VFS reprezentuje jeden konkrétní systém souborů jako takový, zatímco datová struktura VNODE je zobecněním struktury INODE, a reprezentuje jeden konkrétní soubor či adresář. Struktura VFS je vytvářena v rámci připojení souborového systému pomocí operace **mount** (viz 74. díl). Současně s tím je vytvářena i jedna struktura VNODE, reprezentující uzel adresářového stromu, ke kterému je dotyčný souborový systém připojován (viz obr. 77.2.). Další struktury VNODE jsou pak vytvářeny při otevírání jednotlivých souborů (a adresářů) v rámci příslušného souborového systému, a zařazovány do spojového seznamu struktur VNODE, sdruženého s příslušnou strukturou VFS - viz opět obrázek 77.2.

.PI OBR77_2.TIF, 20, 40, 10

Obr. 77.2.: Vztah datových struktur VFS a VNODE

RNODE vs. INODE

Datovou strukturu VNODE je nejlépe chápat jako společné "zastřešení" datových struktur, které již jsou závislé na konkrétní implementaci. Za tímto účelem obsahuje struktura VNODE ukazatel na datovou strukturu, která příslušný soubor či adresář skutečně reprezentuje, a se kterou jsou také sdruženy výkonné procedury, zajišťující nejrůznější akce s příslušným souborem či adresářem. V případě místního souboru je takovouto strukturou původní datová struktura INODE (a ukazatel v rámci struktury VNODE pak ukazuje na exemplář struktury INODE), se kterou jsou sdruženy procedury pro práci s místními soubory.

Pro potřeby reprezentace vzdálených souborů, sdílených prostřednictvím protokolu NFS, byl navržen další druh datové struktury: **RNODE**. Jednotlivé exempláře této datové struktury vznikají pouze na straně klienta, a obsahují mimo jiné i systémové identifikace vzdálených souborů, které si klient vyžádal na příslušném serveru (viz

minulý díl). S těmito vzdálenými soubory se pak pracuje prostřednictvím procedur, které jsou s datovými strukturami sdruženy - a právě tyto procedury implementují vlastní protokol NFS pro sdílení vzdálených souborů v sítích. Přesný způsob fungování těchto konkrétních procedur je ovšem natolik zajímavý, že se mu budeme podrobněji věnovat v samostatném dílu tohoto seriálu.

.PI OBR77_3.TIF, 20, 40, 10

Obr. 77.3.: Představa zpracování požadavku na přístup k souboru

Prozatím si pouze naznačme, jakým způsobem jsou tyto procedury aktivovány. Na straně klienta je prvotním iniciátorem systémové volání, generované aplikační úlohou. Toto volání specifikuje určitou strukturu VNODE, která se zase odkazuje na některou strukturu INODE či RNODE (nebo jiný druh datové struktury, vytvořené pro potřeby konkrétní implementace jiného druhu souborů a adresářů). Podle toho, kam ukazatel v rámci struktury VNODE ukazuje, je pak možné určit konkrétní procedury, které mají být zavolány - viz obrázek 77.3. Výsledkem spuštění procedur, sdružených s datovou strukturou INODE, je pak formulování konkrétního požadavku na server a jeho odeslání.

.PI OBR77_4.TIF, 20, 40, 10

Obr. 77.4.: Rozhraní VFS/VNODE, INODE a RNODE na straně klienta i serveru

Na straně serveru je příjemcem tohoto požadavku systémový proces (tzv. NFS démon). Ten pak na základě přijatého požadavku generuje systémové volání, které opět specifikuje určitý uzel VNODE. Ten ukazuje na konkrétní datovou strukturu, reprezentující požadovaný soubor či adresář (celou situaci názorně ilustruje obrázek 77.4). V této souvislosti je vhodné si zdůraznit, že tentokráte již musí jít o místní soubor či adresář (a nikoli vzdálený). Tvůrci protokolu NFS totiž vcelku rozumně zakázali tranzitivnost sdílení souborů v sítí. Tedy takovou situaci, kdy server nabízí svým klientům soubory, které sám získává v roli klienta od jiného serveru. Ačkoli by to bylo principiálně možné, není to efektivní. Místo přes prostředníka se totiž každý klient může obrátit přímo na ten server, který požadované soubory skutečně vlastní (jako lokální).

BSD Unix vs. AT&T Unix

Rozhraní VFS/VNODE, které jsme si až dosud popisovali, bylo vytvořeno v rámci tzv. BSD větve Unixu verze 4.2 (viz 75. díl seriálu), kde se také protokol NFS prosadil nejdříve. V druhé hlavní větvi, tzv. AT&T Unixu, existovalo ve verzi System V Release 3 rozhraní s velmi podobnými datovými strukturami FSS (File System Switch). Bylo vyvinuto pro potřeby protokolu RFS (Remote File Sharing) firmy AT&T, který má stejné poslání jako protokol NFS - tedy zajišťovat transparentní sdílení souborů v sítích (ovšem na rozdíl od bezestavového protokolu NFS funguje jako stavový). Protokol NFS se však ukázal být životaschopnějším, a jeho používání se posléze prosadilo i do AT&T Unixu. Spolu s ním pak i rozhraní VFS/VNODE, které se stalo standardní součástí AT&T Unixu od verze System V Release 4.

78/ RPC I.

V minulých třech dílech jsme se podrobněji zabývali celkovou filosofií i některými implementačními aspekty protokolu NFS, který má v rodině protokolů TCP/IP na starosti plně transparentní sdílení souborů v prostředí počítačových sítí. Kromě své samotné podstaty je ale tento protokol zajímavý také tím, že pro jeho praktickou implementaci byl poprvé ve významnějším měřítku použit mechanismus, označovaný jako RPC (Remote Procedure Call). Dnes se podrobněji seznámíme s celkovou filosofií a základními aspekty tohoto mechanismu, a v příštím dílu si ukážeme, jak je konkrétně použit pro implementaci protokolu NFS v prostředí Unixu.

Nejprve si ale zopakujme, jakým principiálním způsobem je v protokolu NFS řešen požadavek klienta na přístup ke vzdálenému souboru: k uspokojení tohoto požadavku je nutné provedení určitých akcí (např. čtení či zápisu na disk), které ale musí proběhnout na tom počítači, na kterém se soubor skutečně nachází - tedy na počítači v roli serveru. Na straně klienta, který o přístup ke vzdálenému souboru žádá, je proto zformulován požadavek na provedení těchto akcí, je sestaven ve formě zprávy, a tato je odeslána serveru. Ten zprávu přijme, provede požadované akce, a jejich výsledek vrátí zpět klientovi jako svou odpověď.

Nyní si zkusme představit, jakým konkrétním způsobem může být toto obecné schéma realizováno. Aplikace, která požadavek na přístup ke vzdálenému souboru vznáší, si vzhledem k plně transparentnímu sdílení nemusí vůbec uvědomovat, že jde o vzdálený soubor - svou žádost tedy formuluje přesně stejně, jako kdyby šlo o místní soubor (v prostředí Unixu formou systémového volání, viz minulý díl). Skutečnost, že jde o vzdálený soubor, si plně uvědomuje až ta programová entita, která příslušný požadavek skutečně vyřizuje. Ta si také musí být vědoma, že pracuje v prostředí sítě, musí si uvědomovat existenci vzdálených uzlů a musí znát způsob, jak s nimi komunikovat. Tato konkrétní programová entita, kterou ve víceúlohovém prostředí bude nejspíše samostatný proces, tedy nejprve sestaví příslušný požadavek ve formě zprávy, tu odešle, a pak čeká na odpověď serveru. Toto čekání přitom bude nejspíše realizováno jako suspendování příslušného procesu (tedy jeho převedení ze stavu "probíhající" do stavu "čekající"), s požadavkem na následné aktivování v okamžiku příchodu odpovědi (která bude zřejmě signalizována prostřednictvím mechanismu přerušení).

Zkusme si právě popsaný postup zrekapitulovat: proces, který dostal za úkol uspokojit požadavek na přístup ke vzdálenému souboru, zajistí provedení příslušných akcí prostřednictvím akcí typu odeslání zprávy a čekání na vnější událost (která je z jeho pohledu asynchronní).

Tento postup je samozřejmě možný, a v praxi je také hojně používán. Je ovšem v zásadním rozporu se současnou představou o tom, jak správně psát programy - tedy se zásadami strukturovaného programování. Vzhledem k tomu pak neumožňuje nasadit osvědčené metody návrhu a vývoje rozsáhlých programových celků, které jsou na strukturovaném programování založeny.

Představa akce, která je iniciována vysláním zprávy, probíhá asynchronně (nezávisle na dalším průběhu výpočtu), a její konec je signalizován přerušením právě

probíhajícího programu, skutečně vůbec nezapadá do rámce strukturovaného programování - to se naopak snaží dívat na každou výkonnou akci jako na proceduru, která se začne provádět v okamžiku jejího zavolání a končí návratem z této procedury, neboli předáním řízení bezprostředně za místo jejího volání - viz obrázek 78.1.

.PI OBR78_1.TIF, 20, 40, 10

Obr. 78.1.: Představa volání lokální a vzdálené procedury

Tomu, aby se tato představa mohla aplikovat i na zajištění přístupu ke vzdáleným souborům, stojí v cestě jedna významná skutečnost - vlastní výkonné akce nebudou probíhat na tom počítači, na kterém je požadavek na jejich provedení vznesen. Pokud bychom tedy chtěli vyhovět zásadám strukturovaného programování, potřebovali bychom nějaký mechanismus, který by nám umožnil volat procedury na jednom uzlovém počítači, ale skutečně je provádět na jiném uzlovém počítači. Tedy volat takové procedury, které jsou z pohledu volajícího **vzdálenými procedurami** (**remote procedures**). Představu volání takovéto vzdálené procedury ukazuje opět obrázek 78.1.: požadavek na provedení vzdálené procedury nechť vznáší programová entita (proces A) na straně klienta, a to obvyklou formou volání procedury. Tato je ovšem vzdálená, proto je její skutečné provádění zahájeno na vzdáleném uzlu (serveru). Když provádění této procedury skončí, dojde i na straně klienta k předání řízení bezprostředně za místo volání vzdálené procedury (tedy k běžnému návratu z volané procedury).

Mechanismus, který právě naznačený způsob volání vzdálených procedur umožňuje, se pak příznačně označuje jako **RPC** (**Remote Procedure Call**, doslova: **volání vzdálených procedur**). V ideálním případě zcela zakrývá jakýkoli rozdíl mezi voláním místní a vzdálené procedury, takže volající si ani nemusí být vědom, že jím volaná procedura se ve skutečnosti provádí na vzdáleném počítači (dosažení tohoto ideálního stavu ale stojí v cestě některé technické problémy, o kterých se zmíníme později).

Zastavme se nyní u toho, v čem tkví skutečná odlišnost vzdáleného volání procedur od původní představy explicitního zasílání zpráv a čekání na odpovědi. Samotná komunikace mezi dvěma uzly sítě bude vždy muset mít formu předávání zpráv - rozdíl je zde pouze v tom, kdo a jak bude tyto zprávy sestavovat a odesílat, a také čekat na odpovědi a vyhodnocovat je. Vžijme se do postavení toho, kdo píše programovou entitu, skutečně zajišťující přístup ke vzdáleným souborům - v souladu s obrázkem 78.1. proces A. Bez mechanismu RPC musí proces A sám explicitně sestavovat a odesílat zprávy, vhodným způsobem čekat na odpovědi a tyto pak vyhodnocovat. Ten, kdo píše (a hlavně ladí) zdrojový tvar tohoto procesu, se pak musí obejít bez všech podpůrných prostředků, které pro vývoj a tvorbu strukturovaných programů existují.

Naproti tomu při existenci mechanismu RPC je vlastní komunikace se vzdáleným uzlem před procesem A skryta - sestavování a odesílání zpráv i čekání na odpovědi je zde realizováno v rámci programových entit, které implementují mechanismus RPC. Tyto entity pak vůči procesu A vystupují jako lokální procedury, které proces A může obvyklým způsobem volat. Konkrétní představu ilustruje obrázek 78.2.: proces A,

který potřebuje zajistit provedení určitých akcí na vzdáleném uzlu (serveru), pouze volá příslušnou lokální proceduru, která je součástí implementace mechanismu RPC (a označuje se jako **stub**, v češtině pak: **spojka**). Tato spojka (procedura) zajistí vše potřebné (včetně čekání na příchod odpovědi), a poté řádným způsobem skončí, neboli vrátí řízení zpět procesu A, bezprostředně za místo svého volání.

Pokud bychom tedy vše maximálně zjednodušili, mohli bychom se na mechanismus RPC dívat jako na "vrstvičku", která překrývá explicitní komunikaci se vzdáleným uzlem (založenou na předávání zpráv), a nahrazuje ji voláním lokálních procedur (spojek).

Ve skutečnosti je ovšem úloha mechanismu RPC mnohem obecnější. Kromě příslušných spojek na straně klienta tento mechanismus definuje i obdobné spojky na straně serveru, které přijímají zprávy od spojek klientů, a na jejich základě pak volají výkonné procedury, které zajistí provedení požadovaných akcí - tedy procedury, které jsou z pohledu klienta (procesu A) vzdálené, ale pro spojku na straně serveru již jsou lokální! Dále je součástí definice mechanismu RPC i přesný způsob komunikace mezi spojkami klientů a serverů, a v neposlední řadě i repertoár vzdálených procedur, možnosti a způsoby rozšiřování tohoto repertoáru atd.

.PI OBR78_2.TIF, 20, 40, 10

Obr. 78.2.: Představa spojek (stubs)

Zastavme se ale ještě u některých technických aspektů mechanismu RPC, které nám umožní lépe pochopit jeho samotnou podstatu. V souladu s obrázkem 78.2. vycházejme opět z toho, že tím, kdo na straně klienta mechanismus RPC bezprostředně využívá (aniž si to nutně musí uvědomovat), je proces A. Pokud například potřebuje načíst část souboru, který se nachází na vzdáleném počítači, bude za tímto účelem volat proceduru (např. *read*), která je ve skutečnosti spojkou (stub). Této spojce-proceduře přitom předá všechny potřebné parametry stejným způsobem, jako kterékoli jiné lokální proceduře. Spojka pak na základě svého volání sestaví zprávu pro svou partnerskou spojku na straně serveru, a v ní mj. uvede, která vzdálená procedura má být provedena. Parametry, které spojka klienta dostala při svém volání, však ve skutečnosti "patří" vzdálené proceduře. Spojka klienta je proto převede do takového tvaru, který je vhodný pro přenos (tomuto úkonu se říká **marshalling**, nebo též: **serializing**), a připojí je ke zprávě, odesílané spojce serveru. Tato spojka pak zprávu přijme, v ní obsažené parametry "rozbalí" (provede tzv. **unmarshalling**, též: **deserializing**), a zajistí volání požadované procedury. Tato je pro spojku na straně serveru lokální procedurou, a proto ji všechny potřebné parametry předá způsobem, obvyklým pro lokální procedury. Jakmile výkonná procedura skončí, vrátí řízení tomu, kdo ji volal - tedy spojce serveru. Ta vezme všechny případné výstupy, upraví je do vhodného tvaru pro přenos, a odešle zpět spojce klienta.

S předáváním parametrů je ovšem spojena hned celá řada technických problémů. Nejjednodušší je situace v případě, kdy jsou parametry vzdálené procedury volány hodnotou. Pak totiž stačí vytvořit jejich kopii, tu odeslat spojce serveru, a při skutečném volání vzdálené procedury je opět předat jako parametry volané hodnotou.

V případě volání referencí (neboli: odkazem) dostává volaná procedura na straně klienta (tj. spojka) pouze ukazatel na objekt, který je jejím skutečným parametrem. Tento objekt však existuje jen na straně klienta, a proto není možné použít tentýž ukazatel i na straně serveru a předat jej při skutečném volání vzdálené proceduře. Možným řešením je v tomto případě strategie *copy/restore*: ta předpokládá, že objekt, na který ukazatel ukazuje, je nejprve zkopírován (resp. přenesen) i na server. Vzdálená procedura pak při svém volání dostane ukazatel na tento zkopírovaný exemplář programového objektu, který pak může v rámci své činnosti příslušným způsobem modifikovat. Jakmile provádění vzdálené procedury skončí, je dotyčný objekt zase zkopírován zpět na klienta.

Další okruh problémů je pak spojen s tím, že různé uzlové počítače mohou používat odlišné konvence pro reprezentaci nejrůznějších operandů, které jsou předávány v roli parametrů vzdálených procedur. Proto je v rámci "balení" parametrů před přenosem (marshalling, serializing) a při následném "rozbalení" třeba provádět i potřebné konverze. O tom, jak je tato otázka řešena v prostředí TCP/IP sítí, si ale povíme v dalších pokračováních.

79/ RPC II.

V minulém dílu jsme se začali podrobněji zabývat mechanismem volání vzdálených procedur (RPC, Remote Procedure Call), a to v kontextu implementace protokolu NFS pro transparentní sdílení souborů v sítích. Tento všeobecně použitelný mechanismus byl totiž poprvé ve významnějším měřítku použit právě pro implementaci protokolu NFS. Minule jsme se seznámili s jeho celkovou filosofií, a dnes již dojde řada na to, jakým konkrétním způsobem tento obecný mechanismus implementovala a standardizovala firma Sun Microsystems.

Úvodem je vhodné si zdůraznit vztah firmy Sun Microsystems k mechanismu RPC a jeho standardizaci - platí zde přesně totéž, co jsme si v 75. dílu říkali již v souvislosti s protokolem NFS. Samotné volání vzdálených procedur můžeme chápat jako obecnou myšlenku, postup či techniku, kterou může kdokoli implementovat podle svých vlastních představ. Firma Sun Microsystems tak učinila, své konkrétní řešení zveřejnila, toto se ujalo a stalo se všeobecně uznávaným standardem v rámci rodiny protokolů TCP/IP (kodifikovaným ve formě dokumentu RFC, viz 75. díl). Pokud se tedy v dalším budeme odvolávat na mechanismus RPC resp. na standard RPC, budeme tím mít na mysli jednu konkrétní implementaci, resp. jeden konkrétní standard, pocházející od firmy Sun Microsystems.

Identifikátory vzdálených procedur

Pro správné pochopení způsobu, jakým je mechanismu RPC implementován, je vhodné začít s následující představou:

každá vzdálená procedura má přiřazen jednoznačný číselný identifikátor, a její vzdálené volání (tj. volání na straně klienta) má obecně tvar

CALL (<číslo_vzdálené_procedury>)

Praktická realizace této jednoduché myšlenky ovšem vyžaduje zavést vhodný řád do přidělování takovýchto číselných identifikátorů - aby byla zachována konzistence číslování procedur a jejich identifikátory byly skutečně jednoznačné (tedy aby se

nikdy nemohlo stát, že dvě různé vzdálené procedury dostanou přiděleny stejné identifikátory). K tomu je ovšem nutná existence jediného centrálního subjektu, který bude přidělování těchto identifikátorů vhodně koordinovat. Této role se podujala právě firma Sun Microsystems.

Aby firma Sun Microsystems nemusela přidělovat jednoznačný identifikátor pro každou jednotlivou vzdálenou proceduru (což by bylo organizačně neúnosné), rozhodla se pro použití takových identifikátorů, které se skládají ze tří složek:

.cp20

- z tzv. čísla programu (**program number**), které souhrnně identifikuje skupinu procedur, zajišťujících určitou službu. Například všechny vzdálené procedury, které jsou používány v rámci implementace protokolu NFS, tvoří jednu takovouto skupinu, a mají tudíž přiřazeno jedno číslo programu (pro NFS konkrétně 10003).

- z čísla procedury (**procedure number)**, které jednoznačně identifikuje příslušnou proceduru v rámci její skupiny, a

- z čísla verze (**version number**).

Logika, která stojí za tímto rozdělením identifikátorů na tři složky, je vcelku zřejmá: firmě Sun jako globálnímu koordinátorovi stačí pečovat pouze o jednoznačnost první složky. Kdokoli, kdo se rozhodne implementovat pomocí mechanismu RPC nějakou novou službu, si od firmy Sun může vyžádat jednoznačné číslo programu. Jednotlivým procedurám, které pak pro zajištění své služby vytvoří, již přiděluje druhou složku (číslo procedury) sám. Konečně třetí složka vychází vstříc postupnému vývoji jednotlivých služeb, který dává postupně vznikat novým a novým verzím. Díky této složce identifikátoru vzdálené procedury je pak možné průběžně implementovat nejnovější verze, ale současně s tím zabezpečit i zpětnou kompatibilitu a podporovat i verze předchozí.

Jeden parametr stačí

V zájmu snazší implementace zavedla firma Sun konvenci, že všechny vzdálené procedury mají právě jeden vstupní parametr, a právě jeden výstupní parametr (resp. vrací jediný výstup). Není ale tato konvence příliš omezující? Nikoli - pokud by bylo zapotřebí předat více parametrů, tyto se vhodně "zabalí" (přesněji: vytvoří se z nich vhodná datová struktura), a jako jediný vstupní parametr bude vzdálené proceduře předán ukazatel na tuto datovou strukturu, která samozřejmě musí být v rámci vzdáleného volání přenesena na server. K jejímu správnému využití je dále nutné, aby obě strany (tj. klient i server) byly předem dohodnuty na formátu této datové struktury a významu jejích jednotlivých částí. To se ale dá vcelku snadno zařídit (viz dále).

XDR - eXternal Data Representation

Poněkud složitější je ale to, aby obě strany také správně interpretovaly každou jednotlivou část vstupních a výstupních dat vzdálených procedur. Budou-li například srozuměny s tím, že obsah dvou bytů má představovat celé číslo bez znaménka, mohou jej stále ještě interpretovat různě - jedna strana může považovat za vyšší ten z obou bytů, který druhá strana naopak považuje za nižší.

Právě naznačený problém správné interpretace přenášených dat má dvě principiální řešení: první spočívá v tom, že každá zúčastněná strana bude předem znát konvence, které používá kterákoli druhá strana. Pak je možné při přenosu dat provést potřebné

konverze nejvýše jednou - ať již u odesilatele, nebo u příjemce (případně je vůbec neprovádět, pokud obě strany používají přesně stejné konvence). Alternativním řešením je zavést jeden společný mezitvar, a veškerá data pak vždy přenášet v něm. To sice znamená provádět nezbytné konverze dvakrát (na straně příjemce i na straně odesilatele), ale současně to znamená i to, že každá strana vystačí vždy jen s jednou sadou konverzních prostředků, a nemusí se jakkoli přizpůsobovat případným novým konvencím druhých stran. No a právě toto druhé řešení zvolila firma Sun pro implementaci svého mechanismu RPC.

Konkrétní realizací této volby je pak standard XDR (eXternal Data Representation), který byl opět zveřejněn, je všeobecně uznáván jako standard v rámci rodiny protokolů TCP/IP, a je kodifikován formou dokumentu RFC.

Standard XDR tedy definuje jednotný způsob reprezentace přenášených dat, nezávislý na konkrétní architektuře jejich odesilatele i příjemce. Kromě toho je součástí XDR také jazyk pro nezávislý popis těchto dat. Po stránce implementační souvisí s tímto standardem také konkrétní konverzní rutiny, které mají nejčastěji formu knihovních rutin, a jsou obvykle označovány jak XDR filtry.

Volání vzdálených procedur na straně klienta

Nyní se můžeme opět vrátit k naší první představě volání vzdálených procedur - jakmile je dokážeme jednoznačně identifikovat pomocí vhodného identifikátoru (složeného z čísla programu, čísla verze a čísla procedury), může nám k jejich volání (na straně klienta) stačit vlastně jediný prostředek - systémová rutina, pojmenovaná příznačně **callrpc**. Ta má celkem osm parametrů:

- identifikaci uzlu, na kterém má být vzdálená procedura provedena
- číslo programu (program number) vzdálené procedury
- číslo verze (version number) vzdálené procedury
- číslo vzdálené procedury v rámci její skupiny (procedure number)
- vstupní parametr vzdálené procedury
- XDR filtr vstupního parametru (který definuje konverzní rutinu pro převod vstupního parametru do přenosového tvaru)
- výstupní parametr vzdálené procedury
- XDR filtr výstupního parametru

Na straně klienta lze vystačit jen s tímto jediným prostředkem, pomocí kterého lze volat libovolnou vzdálenou proceduru. Jak tomu ale bude na straně serveru, kde jsou jsou tyto vzdálené procedury skutečně prováděny?

Registrace vzdálených procedur

Na straně serveru musí vždy existovat někdo, kdo má přehled o službách poskytovaných formou vzdálených procedur - konkrétně o všech vzdálených procedurách, které je možné na daném serveru volat. Tím, kdo tento přehled má, je tzv. RPC dispečer (RPC library dispatcher), který je jednak příjemcem všech žádostí klientů o volání vzdálených procedur, a jednak skutečně volá ty lokální rutiny, které vzdálené procedury implementují (a vůči klientům proto vystupuje v roli spojky serveru, viz minulý díl). Každá lokální procedura, která chce implementovat některou

vzdálenou proceduru, se musí u tohoto dispečera registrovat - přitom mu musí sdělit nejen to, o kterou vzdálenou proceduru se jedná, ale i konkrétní způsob svého volání, což vzhledem ke konvenci o jednom vstupním parametru a jednom výstupním parametru vzdálených procedur znamená v podstatě sdělení o tom, které konverzní rutiny má dispečer použít pro konverzi těchto parametrů z/do přenosového tvaru (neboť to musí být právě RPC dispečer, kdo potřebnou konverzi iniciuje).

Konkrétním prostředkem, kterým se lokální procedura registruje u RPC dispečera, je systémová rutina **registerrpc** s následujícími parametry:

- číslo programu (program number) vzdálené procedury
- číslo verze (version number) vzdálené procedury
- číslo vzdálené procedury v rámci její skupiny (procedure number)
- vstupní bod lokální procedury, která implementuje vzdálenou proceduru
- vstupní parametr vzdálené procedury
- XDR filtr vstupního parametru
- výstupní parametr vzdálené procedury
- XDR filtr výstupního parametru

Tři úrovně RPC

Právě naznačený způsob využití mechanismu RPC (na úrovni systémových rutin **callrpc** a **registerrpc**) není zdaleka jediný možný.

Mechanismus RPC lze obvykle využívat na třech různých úrovních, přičemž ta, kterou jsme až dosud předpokládali, představuje tu prostřední. Je charakteristická tím, že na této úrovni je nutné si uvědomovat existenci distribuovaného prostředí (tj. existenci vzdálených uzlů), a vyžaduje také znalost konkrétních vzdálených procedur (které je nutně explicitně určovat). Vše ostatní je ale podřízeno snaze o maximální jednoduchost využití celého mechanismu volání vzdálených procedur.

To ale nemusí být vždy výhodné. Při seriózní tvorbě aplikací, které mechanismus RPC využívají, může být velmi nevýhodné, že na této úrovni není možné nijak ovlivnit celou řadu konkrétních aspektů - například to, jakým konkrétní transportní mechanismus je využíván pro skutečný přenos v síti (zda jde např. o nespolehlivou datagramovou službu protokolu UDP, nebo o spolehlivou spojovanou službu protokolu TCP), jaké časové limity (timeout-y) jsou používány, jak ještěšena otázka chyb, ověřování přístupových práv a totožnosti (authentication) apod.

Standard RPC je řešen nezávisle na transportním protokolu (aby mohl být implementován nad různými protokoly), a nesnaží se sám zavádět jakoukoli dodatečnou spolehlivost (ošetřováním chyb). Aplikace, která mechanismus RPC využívá, si proto musí uvědomovat, jaký transportní mechanismus je pro implementaci RPC použit, a sama si z toho vyvodit příslušné důsledky (mj. z hlediska spolehlivosti). Pokud potřebuje nějak zasáhnout do způsobu, jakým RPC transportní prostředky využívá, pak k tomu musí využít zmíněnou nejnižší úroveň práce s mechanismem RPC (což v podstatě znamená realizovat prostředky typu **callrpc** pomocí prostředků nižší úrovně).

Práce s mechanismem RPC na této nejnižší úrovni jej již značně netriviální, a je míněna především pro odborníky, kteří vytváří nové systémové prostředky a služby. Jejich náročný úkol přitom mohou usnadnit různé nástroje, mezi které patří zejména překladač **rpcgen**, vyvinutý firmou Sun. Jeho hlavním úkolem je překlenout rozdíl mezi střední a nejnižší úrovní práce s mechanismem RPC, zbavit programátory maxima "špinavé práce", a umožnit jim soustředit se na to, co je pro jejich aplikaci podstatné. Na základě obecnějšího popisu ve zvláštním RPC jazyku (velmi blízkému k jazyku C) totiž překladač **rpcgen** generuje to, co by jinak bylo třeba explicitně naprogramovat na nejnižší úrovni (ve formě zdrojových textů jazyka C).

Naproti tomu na nejvyšší úrovni je celý mechanismus RPC obvykle "zabalen" takovým způsobem, že jeho samotná podstata již nemusí být vůbec patrná. Prostředky, které jsou na této úrovni k dispozici, již nejsou typu "volej tu a tu vzdálenou proceduru", a vůbec nepracují s čísly programů, procedur a verzí. Místo toho jde o lokální procedury, které vesměs přímo odpovídají jednotlivým vzdáleným procedurám (stylem 1:1), a také již nepotřebují dodržovat konvence o jediném vstupním a jediném výstupním parametru. Mají formu zdrojových knihoven, a mohou být přímo začleněny do zdrojových tvarů nejrůznějších aplikací, psaných ve vyšších programovacích jazycích (například v jazyku C). Vzhledem k tomu je pak tato nejvyšší úroveň určena pro méně náročné aplikační programování, které je ale možné prakticky i bez jakéhokoli tušení o existenci mechanismu RPC.

80/ Elektronická pošta - I.

Zdaleka nejpopulárnější službou dnešních počítačových sítí je elektronická pošta, které "přichází na chuť" stále více a více uživatelů. Přestože jde o službu, která může být v sítích různého typu implementována dosti odlišným způsobem, její celkový efekt je pro uživatele prakticky vždy stejný. Dnes (a ještě příště) se proto zaměříme právě na tento uživatelský pohled na elektronickou poštu - na její celkovou filosofii a principiální možnosti. V dalších dílech se pak podrobněji seznámíme s tím, jak je přenos zpráv v rámci elektronické pošty řešen v sítích na bázi TCP/IP

Elektronickou poštu je možné bez nadsázky označit za novodobý fenomén - zcela zásadním způsobem totiž mění způsob, jakým spolu lidé komunikují. Jestliže například v 18. století trvalo několik týdnů, než se nějaká zpráva dostala z Evropy do Ameriky či naopak, koncem minulého století se to stihlo již do minuty, zatímco dnes se přenos zpráv i na druhou stranu zeměkoule měří spíše na sekundy, než na minuty. Díky elektronické poště a dalším moderním způsobům komunikace se dnešní svět skutečně stává tím, čemu se v angličtině říká "global village" (a v češtině nejspíše: jedna velká vesnice). Co se kde jen "šustne", to se okamžitě ví po celém světě.

Základem jsou poštovní schránky

Pro správné pochopení podstaty elektronické pošty je vhodné využít analogie s běžnou listovní poštou a představovat si, že každý jednotlivý účastník je vybaven vlastní poštovní schránkou, do které mu jsou doručovány jednotlivé dopisy (zprávy). Dále předpokládejme, že tyto schránky nejsou umístěny přímo v domech, kde lidé bydlí, ale pouze na poštovních úřadech (v angličtině se těmto schránkám na poštovních úřadech říká Post Office Box, zkratkou P.O.Box či jen P.O.B.). Adresa

takovéto poštovní schránky je pak dána dvojicí údajů: adresou poštovního úřadu, a číslem schránky v rámci tohoto úřadu - tedy například: P.O.Box 123, Pošta Praha 5.

V případě elektronické pošty je rozdíl v tom, že roli poštovního úřadu hraje uzlový počítač sítě, a poštovní schránky jsou místo skutečných přihrádek pouze vymezená místa na disku. Adresa takovéto poštovní schránky, označované také jako **mailbox**, je opět tvořena dvojicí údajů: adresou počítače, a identifikátorem (označením) schránky v rámci tohoto počítače.

Podobně jako u běžné listovní pošty (s výjimkou zásilek, určených do vlastních rukou), platí i zde zásada: jednotlivé zprávy jsou doručovány do poštovních schránek - dále již záleží na uživateli, kdy se do své schránky podívá, a poštu v ní obsaženou zpracuje. V silách dnešní elektronické pošty je doručit zprávu (tj. uložit ji do poštovní schránky jejího adresáta) třeba až na druhou stranu zeměkoule za několik málo sekund. Není už ale v silách elektronické pošty přinutit uživatele, aby se do své poštovní schránky skutečně podíval - to může trvat i celé hodiny a dny.

Adresy, používané pro elektronickou poštu, jsou tedy v zásadě adresami poštovních schránek, a nikoli adresami uživatelů jako takových. Elektronická pošta je doručována "majiteli schránky XY", a nikoli "uživateli XY". Díky tomu je pak v principu možné zřizovat i takové poštovní schránky, které patří celým skupinám uživatelů (kteří sdílí jednu schránku, např. kvůli omezené kapacitě disku), a na druhé straně i schránky, které nepatří žádným "skutečným" uživatelům (příkladem může být schránka, dočasně zřízená pro přijímání přihlášek na nějakou konferenci apod.).

Existují ovšem i významné výjimky z tohoto obecného principu: adresy, používané v rámci standardu X.400 pro elektronickou poštu v rámci referenčního modelu ISO/OSI (viz 38.díl), specifikují právě konkrétního člověka (jeho jménem, příjmením, příslušností k určitému organizačnímu útvaru apod.), a naopak explicitně nedefinují jeho poštovní schránku ani její umístění.

Adresy pro elektronickou poštu

Přesná pravidla, podle kterých jsou obě složky elektronických adres sestavovány do jediné výsledné adresy, se v jednotlivých sítích více či méně liší. Nejčastěji mají výsledné adresy tvar

schránka@počítač

ovšem existují i četné jiné způsoby sestavování, které se z tohoto schématu zcela vymykají (ze starších například konvence o adresováních v sítích na bázi UUCP, a z novějších pak způsob adresování, požadovaný standardem X.400).

Jméno (resp. identifikátor) schránky je nejčastěji shodné se jménem toho, komu patří - přesněji se jménem, pod jakým operační systém daného počítače "zná" příslušného uživatele. Zde je dobré si uvědomit, že jednotliví uživatelé musí mít ve víceuživatelských operačních systémech zřízeny tzv. uživatelské účty (user accounts), v rámci kterých jim jsou vymezena nejrůznější přístupová a další práva, a obvykle také i přidělena schránka pro elektronickou poštu. Tyto uživatelské účty přitom musí být nějak pojmenovány - a právě jejich jména pak slouží současně i jako jména poštovních schránek.

Jména počítačů, která tvoří druhou část elektronických adres, mohou být "jednorozměrná" (tedy tvořená jediným, dále nestrukturovaným jménem, jako

například v síti Bitnet), nebo mohou být členěna do hierarchicky uspořádaných domén a subdomén (kterými jsme se zabývali již v 51. dílu seriálu). Záleží přitom na konvenci sítě, do které je příslušný počítač zapojen jako jeden z jejích uzlů.

Například autor tohoto článku má na počítači, zapojeném do sítě BITNET (s "jednorozměrným" jménem CSPGUK11, jaké tato síť používá) uživatelský účet se jménem PETERKA. Jeho adresa pro příjem elektronické pošty na tomto počítači je proto

PETERKA@CSPGUK11

Na jiném počítači, zapojeném do sítě Internet (a s doménovým jménem KKI.MS.MFF.CUNI.CZ) má uživatelský účet se stejným jménem, a pro příjem elektronické pošty na tomto počítači tak má adresu

PETERKA@KKI.MS.MFF.CUNI.CZ

Ještě na jiném počítači (FRODE.DCIT.CZ) pak má uživatelský účet se jménem PET, a tudíž adresu pro elektronickou poštu

PET@FRODE.DCIT.CZ

Všechno může být jinak

V praxi ovšem mohou být elektronické adresy různým způsobem modifikovány. Například když si majitel uživatelského účtu PET na počítači FRODE.DCIT.CZ usmyslí, že PET je příliš krátké a nelíbí se mu, dají se věci zařídit tak, aby mohl přijímat elektronickou poštu například na adrese

PETERKA@FRODE.DCIT.CZ

nebo

JIRI.PETERKA@FRODE.DCIT.CZ

apod. K tomu stačí vlastně jen maličkost: té části operačního systému počítače FRODE.DCIT.CZ, která má na starosti příjem elektronické pošty, se předepíše, že veškerou poštu pro uživatele PETERKA či JIRI.PETERKA má ukládat do poštovní schránky, která přísluší uživatelskému účtu PET. Konkrétní mechanismus, který tohoto efektu dosáhne, si můžeme představit jako zavedení synonym (alias-ů) pro uživatelský účet PET.

Další široký prostor pro modifikaci adres pro elektronickou poštu skýtají jejich druhé části, představující jméno uzlového počítače. Zde je ovšem situace velmi závislá na konkrétní síti. Například v síti Internet, kde se používají doménová jména (odpovídající hierarchicky strukturovaným doménám, viz 51. díl), je v některých případech možné vynechávat určité části těchto jmen (konkrétně ty, které v hierarchickém uspořádání domén odpovídají nejnižším vrstvám).

Představme si následující příklad: organizace DCIT má zřízenu stejnojmennou subdoménu (neboli doménu druhé úrovně) pod doménou CZ (odpovídající České republice jako takové), neboli doménu DCIT.CZ. Do své lokální sítě pak má zapojeny různé uzlové počítače, které také mají svá "místní" jména. Při tvorbě doménových jmen těchto konkrétních počítačů se pak jejich "místní" jména přidávají zleva (v roli domén nejnižší úrovně) k doménám vyšších úrovní. Například:

uzlový počítač FRODE má doménové jméno FRODE.DCIT.CZ,

uzlový počítač EINAR má doménové jméno EINAR.DCIT.CZ

apod.

Při sestavování adres pro elektronickou poštu je samozřejmě možné použít "úplné" doménové jméno konkrétního počítače, na který má být pošta doručena (např.: PET@FRODE.DCIT.CZ). Je ovšem možné dosáhnout i toho, aby doménové jméno nemuselo být uváděno celé, a jméno konkrétního počítače v něm mohlo chybět (a eventuelně i některé domény nižších úrovní) - tedy aby elektronická pošta správně došla například i na adresu PET@DCIT.CZ.

V rámci informací, které definují doménu DCIT (jako subdoménu pod doménou CZ), je totiž uvedeno mj. i to, kam má být doručována elektronická pošta, adresovaná doméně DCIT jako takové (tedy nespecifikující konkrétní počítač v rámci domény DCIT). V našem konkrétním případě nechť je takto stanoveno, že má být doručována na počítač EINAR (tj. na počítač s doménovým jménem EINAR.DCIT.CZ). Potom je vlastně adresa PET@DCIT.CZ ekvivalentní adrese PET@EINAR.DCIT.CZ, a elektronická pošta, zasílaná na adresu PET@DCIT.CZ, je tudíž doručována na počítač EINAR.DCIT.CZ.

Uživateli PET ovšem nemusí vyhovovat to, aby jeho elektronická pošta byla doručována právě na počítač EINAR (například již jen proto, že na tomto počítači nemá uživatelský účet, a tudíž ani poštovní schránku). I zde je však snadná pomoc - dá se totiž zařídit (vhodným příkazem), aby veškerou poštu, určenou uživateli PET, počítač EINAR automaticky předával jinému počítači (například právě počítači FRODE). Pošta, zasílaná na adresu PET@DCIT.CZ, je pak ve skutečnosti doručována na počítač FRODE.DCIT.CZ (tedy na stejné místo, jako kdyby byla opatřena "úplnou" adresou PET@FRODE.DCIT.CZ).

Celou situaci s přenosem pošty na různé adresy (ale témuž adresátovi) názorně ilustruje obrázek 80.1.

K čemu je to dobré?

Jaká je ale logika, která stojí za možnostmi zkracování doménových jmen v adresách? Je to vůbec k něčemu dobré?

Odpověď je samozřejmě kladná: představte si například, že v organizaci DCIT dojde časem k obnově strojového parku, a počítač se jménem FRODE zde již nebude existovat (nebo se porouchá, bude odstaven apod.). Uživatel PET tak bude nucen přijímat svou poštu na jiném počítači - například na počítači se jménem KAARE. Jeho nová "úplná" adresa pak bude: PET@KAARE.DCIT.CZ.

Změnu své adresy pak ale uživatel PET musí sám oznámit všem, kteří s ním komunikují prostřednictvím elektronické pošty, a kterým někdy dříve poslal nyní již neplatnou adresu PET@FRODE.DCIT.CZ. Přitom samozřejmě na někoho zapomene, někdo tuto informaci nevezme na vědomí atd., a výsledkem bude dosti nepříjemný zmatek.

Pokud ale uživatel PET zveřejnil pouze svou "zkrácenou" adresu ve tvaru PET@DCIT.CZ, pak výše uvedená změna může zůstat čistě lokální záležitostí, a navenek se nemusí nijak projevit. Pošta s adresou PET@DCIT.CZ totiž bude stále doručována na ten počítač, který je příjemcem pošty pro doménu DCIT.CZ jako

takovou (tj. na počítač EINAR), a jedinou změnou bude nový příkaz tomuto počítači, specifikující kam má dále předávat poštu pro uživatele PET.

.PI OBR80_1.TIF, 20, 40, 10, 1

Obr. 80.1.: Představa doručování elektronické pošty témuž uživateli na různé adresy

81/ Elektronická pošta II.

V minulém dílu jsme se začali zabývat problematikou elektronické pošty z uživatelského pohledu. Naznačili jsme si, jaký je vztah mezi uživateli a poštovními schránkami, a co vlastně specifikují adresy, používané v rámci elektronické pošty. Dnes si již řekneme něco o tom, jak bývá elektronická pošta implementována.

Pro správné pochopení podstaty a možností elektronické pošty je velmi podstatné uvědomit si následující skutečnost: elektronická pošta není téměř nikdy zajišťována jedním jediným programem - prakticky vždy jde o spolupráci více programů (aplikací, procesů, úloh), které jsou různým způsobem specializovány: některé z nich se starají pouze o vlastní přenos zpráv (a před uživatelem zůstávají obvykle skryty), zatímco jiné zase vědomě "nastavují svou tvář" koncovému uživateli, umožňují mu číst došlé zprávy, vytvářet či editovat nové zprávy apod., ale když mají nějakou zprávu skutečně odeslat, pouze ji předají tomu programu, který má jejich přenos na starosti.

Právě naznačenou dělbu práce ilustruje obrázek 81.1.: celý komplex programů, které ve své vzájemné součinnosti zajišťují uživatelům služby elektronické pošty, se obecně označuje jako **Electronic Mail System,** někdy též: **Message Handling System** (**MHS**). Jeho komponenty se dělí do dvou hlavních skupin: na tzv. **uživatelské složky** (**User Agent, UA**), a **přenosové složky** (**Message Transfer Agent, MTA).**

.PI OBR81_1.TIF, 20, 40, 10

Obr. 81.1.: Komponenty systémů elektronické pošty

(v terminologii standardu X.400)

Přenosové složky a systémy přenosu zpráv

Přenosové složky (Message Transfer Agents) jsou těmy, které mají na starosti vlastní přenos zpráv. Jejich obsahu si nevšímají (kromě případů, kdy zajišťují jejich automatickou konverzi z jednoho znakového kódu do druhého) - zajímá je především adresa příjemce, podle které zprávu doručují. Přenosové složky, které jsou provozovány na jednotlivých uzlových počítačích, přitom musí vzájemně spolupracovat (předávat si zprávy mezi sebou). To ovšem znamená, že musí "hovořit stejnou řečí" - tedy používat stejné protokoly pro přenos zpráv, stejné konvence pro adresování atd.

Vzájemně spolupracující přenosové složky, provozované na jednotlivých uzlových počítačích, vytváří dohromady **systém přenosu zpráv** (též: **Message Transfer System** (**MTS**)). V rámci něj se používají jednotné konvence a protokoly, specifické pro konkrétní "poštovní systém" - své vlastní konvence a protokoly přenosu zpráv mají například různé komerční "pošty" typu cc:Mail či Mail602, i systémy na bázi standardu X.400 či SMTP (o kterých si budeme povídat podrobněji).

Znamená to ale, že mezi dvěma odlišnými poštovními systémy nelze přenášet žádné zprávy? Naštěstí nikoli - není totiž žádným principiálním problémem vytvořit poštovní bránu (e-mail gateway), která bude zajišťovat potřebný přechod.

Uživatelské složky

Existence přenosových složek i celých systémů přenosu zpráv je před běžným uživatelem obvykle skryta. To, s čím uživatel komunikuje, je uživatelská složka (UA, User Agent). Teprve ta vytváří potřebné uživatelské rozhraní, prostřednictvím kterého uživatel může využívat služby elektronické pošty - číst došlé zprávy, rušit je či archivovat, vytvářet a editovat nové zprávy atd. K tomu například musí každá uživatelská složka obsahovat mj. i vhodný editor.

Program, realizující uživatelskou složku, je aplikačním programem, který si uživatel vyvolává až na základě potřeby, a může být provozován i na takovém počítači, který není trvale v provozu (zatímco přenosové složky jsou obvykle provozovány trvale, a vyžadují tudíž počítač, fungující v nepřetržitém režimu). Právě z tohoto důvodu jsou poštovní schránky (mailbox-y) jednotlivých uživatelů většinou realizovány v rámci přenosových složek. Zprávy, které dojdou na adresu určitého uživatele (přesněji: na adresu určité poštovní schránky, viz minule), tak mohou být ukládány do příslušné schránky kdykoli. Uživatel se k nim ale dostane až poté, co si spustí program, realizující uživatelskou složku, a jeho prostřednictvím si zprávy přečte.

Poštovní klient a server

Vztah mezi uživatelskou a přenosovou složkou je do značné míry vztahem typu klient-server: při odesílání zprávy se uživatelská složka postará (v interakci s uživatelem) o její sestavení, ale pak ji předá k odeslání přenosové složce (tj. vyžádá si od ní službu, spočívající v přenosu zprávy). Naopak v okamžiku, kdy se uživatel rozhodne podívat na došlou poštu, obrátí se uživatelská složka na přenosovou se žádostí o poskytnutí obsahu příslušné poštovní schránky.

Z tohoto důvodu je uživatelská složka systému elektronické pošty často označována také jako poštovní klient (mail client), a přenosová složka (resp. počítač, na kterém, je provozována) jako poštovní server (mail server). Terminologie, kterou jsme až dosud používali, je charakteristická spíše pro systémy elektronické pošty na bázi standardu X.400 (zatímco v prostředí TCP/IP sítí se hovoří spíše o poštovních serverech a klientech).

Konkrétní způsob komunikace mezi přenosovou a uživatelskou složkou je samozřejmě pevně dán koncepcí celého systému elektronické pošty (MHS). To ovšem zdaleka neznamená, že by tím byla jednoznačně determinována i "vnější tvář" uživatelské složky (poštovního klienta). Tyto složky mohou být implementovány v mnoha různých podobách - od strohých řádkově orientovaných utilit až po uživatelsky přítulné programy v "okenním" provedení, s nejrůznějšími vymoženostmi. V prostředí počítačů PC tedy může být příslušný klient například programem pro prostředí MS DOSu, aplikací pro MS Windows apod. Každý uživatel si může vybrat, který klient mu bude nejlépe vyhovovat - podmínkou je jenom to, aby se takovýto klient dokázal správně "domluvit" s příslušným poštovním serverem.

.PI OBR81_2.TIF, 20, 40, 10

Obr. 81.2.: Představa uživatelské složky (klienta), provozované na počítači v roli poštovního serveru

Postavení poštovního klienta

Velmi zajímavá je i otázka možného postavení uživatelské složky (poštovního klienta), zvláště pak v prostředí lokálních sítí. Předpokládejme, v souladu s obrázkem 81.2., že přenosová složka je implementována na počítači, který vystupuje v roli poštovního serveru, a obsahuje poštovní schránky jednotlivých uživatelů. Uživatelská složka (poštovní klient) může být programem, který běží právě na tomto počítači - uživatelé, kteří chtějí zpracovávat svou poštu jeho prostřednictvím, pak musí pracovat přímo na tomto počítači (z jeho místních terminálů, případně z jiných počítačů prostřednictvím vzdáleného přihlašování).

Další možností je to, aby poštovní klient byl provozován na jiném počítači, než který vystupuje v roli poštovního serveru. Takovýto program si uživatel může spustit na své pracovní stanici (viz obr. 81.3.), a ponechat na něm, aby si vše potřebné "vykorespondoval" po síti se serverem sám.

Toto druhé řešení má četné výhody: například tu, že uživatel nemusí opouštět své pracoviště, a osobně se obtěžovat až k poštovnímu serveru (nebo využít možnosti vzdáleného přihlašování). Výraznou výhodou bývá i větší komfort, který takovéto řešení může nabídnout: klientské programy, provozované na pracovních stanicích (například v prostředí MS Windows), obvykle nabízí svému uživateli mnohem větší pohodlí a mnohem větší repertoár doplňkových funkcí, než analogické programy, provozované na poštovním serveru (který může být například Unixovským počítačem).

.PI OBR81_3.TIF, 20, 40, 10

Obr. 81.3.: Představa uživatelské složky (klienta),

provozované na pracovní stanici lokální sítě

Vzdálený klient

Poštovní klient, určený k provozování na pracovní stanici v síti, obvykle počítá s tím, že svůj server může kontaktovat kdykoli (a s dostatečnou rychlostí). Jednotlivé zprávy, došlé příslušnému uživateli, proto ponechává v jeho poštovní schránce (tedy na serveru), a odsud si je individuálně "stahuje" až v okamžiku, kdy si to uživatel explicitně vyžádá (např. když si chce přečíst určitou konkrétní zprávu). Pokud je potřebuje někam dočasně uložit (například jako již přečtené, ale dosud nesmazané zprávy), ukládá si je opět na server. To má jednu obrovskou výhodu - fungování klientského programu pak není závislé na tom, na které pracovní stanici je provozován. Uživatel tak může se svou poštou plnohodnotně pracovat z kteréhokoli počítače v dané lokální síti.

.PI OBR81_4.TIF, 20, 40, 10

Obr. 81.4.: Představa vzdáleného poštovního klienta

Alternativním řešením je to, aby si klient v určitý okamžik "stáhl" celý obsah uživatelovy poštovní schránky, a pak se na něj již nemusel obracet s jednotlivými žádostmi o čtení konkrétních zpráv. Současně s tím by si takovýto program také ponechával "u sebe" (tj. na svém lokálním disku) i veškerou rozpracovanou poštu.

Tím se sice ztratí možnost plnohodnotné práce z kterékoli stanice v dané síti, ale na druhé je zase minimalizován přenos dat mezi klientem a serverem - díky jednorázovému a dávkovému charakteru komunikace je pak možné vystačit i s mnohem pomalejšími přenosovými cestami. Natolik pomalejšími, že je možné uvažovat i o provozování poštovního klienta na vzdálených počítačích: takovýto klient se může na svůj poštovní server jednorázově připojit například prostřednictvím komutovaného okruhu veřejné telefonní sítě, "stáhnout si" veškerou došlou poštu, poté se odpojit (zavěsit telefon), a poštu zpracovat již jako samostatně běžící aplikace (tj. v nespřaženém, neboli of-line režimu).

Takovýmto způsobem si uživatel může například ze svého domácího počítače zavolat pro došlou poštu, všechnu si ji přenést k sobě, poté zavěsit, a poštu pak zpracovávat v klidu a pohodlí svého domova (a hlavně bez dlouhého blokování telefonní linky a tomu úměrných poplatků). Když se rozhodne nějakou poštu odeslat, může si ji nejprve připravit do větší dávky, a tu pak analogickým způsobem jednorázově přenést na server k odeslání. V principu je samozřejmě možné i takové řešení, při kterém si uživatel odnese od serveru celou dávku došlé pošty na disketě, doma ji zpracuje, vytvoří novou dávku pošty, určené k odeslání, a tu pak opět po disketě přenese k serveru.

82/ Elektronická pošta III.

V minulých dvou dílech, věnovaných problematice elektronické pošty, jsme se zabývali poštovními schránkami, adresami, a principiálním způsobem přístupu uživatele k elektronické poště. Dnes se podrobněji zastavíme u některých funkčních možností elektronické pošty.

Základní funkcí každého systému elektronické pošty je doručování jednotlivých zpráv od jejich odesilatelů až ke koncovým adresátům. S tím ovšem může souviset i celá řada dalších zajímavých a užitečných funkcí, jako například možnost rozesílání zpráv na více adres, možnost zasílání kopií zpráv na vědomí, možnost potvrzování příjmu, možnost připojování příloh ke zprávám, možnost automatického třídění došlých zpráv, vedení adresářů a archivů zpráv apod. Abychom si všechny tyto zajímavé možnosti mohli popsat a vhodně zasadit do kontextu, je dobré si znovu připomenou to nejdůležitější, o čem jsme si povídali v minulém dílu: totiž že v souvislosti s elektronickou poštou se můžeme setkat se dvěma základními kategoriemi programů - s tzv. přenosovými složkami (MTA, Message Transfer Agents), které zajišťují vlastní přenos zpráv (ale nikoli již komunikaci s uživatelem), a s tzv. uživatelskými složkami (User Agents), které naopak s uživatelem bezprostředně komunikují, umožňují mu číst došlé zprávy, sestavovat nové zprávy atd., ale ke skutečnému odeslání je předávají přenosovým složkámn. Pro naše dnešní povídání je povědomí o existenci těchto dvou kategorií programů podstatné proto, abychom si mohli uvědomit, "kde" jsou různé funkce realizovány, a z toho si pak odvodit i to, co předurčuje jejich dostupnost.

Hlavička a tělo zprávy

Další představa, která nám značně zjednoduší naše povídání, je představa o vnitřním členění zpráv elektronické pošty - tyto mají vždy dvě základní části: **hlavičku** (**header**) a **tělo** (**body**).

Tělo obsahuje vlastní text zprávy, a z pohledu systému elektronické pošty není nijak strukturováno - ani uživatelské složky, ani přenosové složky tuto část zprávy nijak neinterpretují (kromě některých speciálních případů, jako například při potřebě konverze z jedné znakové sady do druhé): interpretace toho, co je obsaženo v těle zprávy, je ponechávána plně na uživatelích.

Naproti tomu obsah hlavičky musí být velmi přesně strukturován - právě zde jsou totiž obsaženy veškeré informace, podle kterých jsou jednotlivé zprávy odesílány, přenášeny a doručovány. Každý systém elektronické pošty (MHS, Message Handling System) musí přesně definovat, co má být v hlavičce obsaženo, a jakým konkrétním způsobem to má být vyjádřeno. To proto, aby si jak uživatelské složky (User Agents), tak i přenosové složky (Message Transfer Agents) dokázaly v hlavičce najít ty informace, které ke své činnosti potřebují.

Údaje, obsažené v hlavičkách zpráv

Jaké konkrétní údaje jsou tedy obsaženy v hlavičkách zpráv? Závisí to samozřejmě na konkrétním systému elektronické pošty a na jím používaném standardu, který přesný formát hlavičky definuje. Prakticky vždy jsou ale v hlavičce obsaženy mimo jiné i následující údaje:

- kdo je odesilatelem zprávy
- kdo má být příjemcem zprávy
- datum a čas odeslání zprávy
- předmět zprávy (anglicky: subject, obvykle v rozsahu jedinéřádky)

Jednu konkrétní možnost naznačuje následující příklad:

From: Jiri Peterka <peterka@kki.ms.mff.cuni.cz>

To: Jan Pavelka <pav@dcit.cz>

Date: Mon, 28 Mar 1994 18:42:01 MET

Subject: Jak se dari?

Většinu položek v hlavičce vyplňuje uživatelská složka (User Agent) na straně odesilatele, podle pokynů, které vhodným způsobem získá od svého uživatele. Při samotném přenosu zpráv však mohou být do hlavičky průběžně přidávány ještě další údaje, ve kterých je zaznamenán průběh přenosu zpráv. Například:

Received: from einar.dcit.cz by frode.dcit.cz (Mercury 1.11);

Tue, 8 Mar 94 11:27:16 CET

Received: from ns.ms.mff.cuni.cz by einar.dcit.cz (AIX 3.2/UCB 5.64/4.07)

id AA13120; Tue, 8 Mar 1994 11:23:46 +0100

---

prosím o šetrné zalámání - buď každá položka jen na jedinou řádku, nebo zachovat výše uvedené rozdělení vždy do dvou řádek

---

Z těchto údajů je pak možné vysledovat skutečný "průchod" zprávy přes různé přenosové složky na mezilehlých uzlových počítačích. Tato možnost však pamatuje

spíše na správce sítí, než na běžné uživatele - těm jejich uživatelská složka (User Agent) hlavičku většinou nezobrazuje (alespoň ne v její skutečné podobě). Místo toho jsou uživatelům předkládány jen některé vybrané údaje (např.: o odesilateli a předmětu zprávy), a to navíc v takové formě, jakou příslušný program, vystupující v roli uživatelské složky, pokládá za nejvhodnější.

Například bezprostředně po "výběru" poštovní schránky může být uživateli nabídnut přehledný seznam došlých zpráv (ve kterém je každé zprávě věnována jedna řádka, a na ní je uvedeno například jen jméno odesilatele a předmět zprávy, případně i datum apod.). Uživatel-příjemce může takovýmto seznamem listovat, a když si nějakou zprávu ze seznamu vybere, poštovní program (User Agent) mu zobrazí její obsah (tj. tělo zprávy).

Rozesílání zpráv na více adres

Jednou z velkých předností elektronické pošty je možnost rozeslání jedné a téže zprávy více příjemcům - kolika, to pro odesilatele vlastně není vůbec relevantní, protože jeho "námaha", spojená s takovýmto úkonem, může být úplně stejná, jako při odesílání zprávy jedinému uživateli.

Možnost rozesílání může být realizována více různými způsoby. Jeden z nich, který je výhodnější spíše pro příležitostné využití, spočívá v tom, že místo jednoho konkrétního příjemce se při sestavování zprávy uvedou dva příjemci, tři, čtyři atd. Toho, kdo by tuto možnost chtěl využívat častěji, by však zřejmě brzy omrzelo opakované ruční rozepisování stejných adres. Program, prostřednictvím kterého takový uživatel své zprávy sestavuje a odesílá, mu může vyjít vstříc a umožnit mu předem sestavit seznam příslušných adresátů (resp. tolik takovýchto distribučních seznamů, kolik potřebuje). Při samotném odesílání pak místo adresáta uživatel uvede jen jméno (identifikátor, označení) distribučního seznamu, a poštovní program (User Agent) se pak již sám postará o potřebné "rozepsání".

Tuto možnost, při které již od odesilatele odchází více jednotlivých zpráv, je třeba odlišit od tzv. elektronických konferencí. Ty jsou založeny na myšlence, že na určitém uzlovém počítači je veden seznam uživatelů (účastníků konference), a tento seznam má vlastní adresu pro elektronickou poštu. Když někdo pošle zprávu (tj. jednu zprávu) na tuto adresu, pak je tato zpráva, obvykle zcela automaticky, rozeslána všem uživatelům, uvedeným na příslušném seznamu.

Potvrzení o příjmu - doporučená elektronická pošta?

Jedním z velmi častých požadavků, které uživatelé vznáší na elektronickou poštu, je možnost získat věrohodnou informaci o tom, že určitá zpráva byla v pořádku doručena.

V principu to není žádný problém: do hlavičky odesílané zprávy se zařadí položka, požadující takovéto potvrzení. Na straně příjemce pak může být na základě tohoto požadavku automaticky vygenerována krátká zpráva o doručení, a odeslána zpět odesilateli původní zprávy.

Otázkou ovšem je, kdy vlastně má být takovéto potvrzení generováno? Tehdy, když je zpráva uložena do poštovní schránky příjemce? Pak nevypovídá nic o tom, kdy je zpráva skutečně přečtena. Nebo má být potvrzení generováno až v okamžiku, kdy si příjemce zprávu skutečně přečte? Pak se ale odesilatel zase nemusí včas dozvědět, že

zpráva byla úspěšně přenesena (resp. že se při přenosu nikde neztratila), a že ji tudíž nemusí posílat znovu.

Některé systémy elektronické pošty řeší toto dilema tak, že zavádí dva druhy potvrzení - potvrzení o uložení zprávy do poštovní schránky příjemce, a potvrzení o jejím skutečném přečtení.

Potvrzení o uložení zprávy přitom může generovat již samotná přenosová složka (Message Transfer Agent), která má přenos zpráv na starosti. Naproti tomu potvrzení o skutečném přečtení může generovat až program, prostřednictvím kterého uživatel zpracovává svou poštu - tedy program, který plní úlohu uživatelské složky (User Agent).

V praxi ovšem nebývá generování takovýchto potvrzení vždy zaručeno. Ne všechny programy, zajišťující fungování elektronické pošty, totiž generování takovýchto potvrzení podporují. Jiné programy zase dávajím uživatelům možnost individuální volby. Pokud tedy někomu pošlete zprávu s požadavkem na potvrzení jejího přijetí (ať již ve smyslu uložení do poštovní schránky či skutečného přečtení), a toto potvrzení nedostanete, ještě nemusí být zdaleka vše špatně!

Kopie na vědomí

V běžném úředním styku, vedeném písemnou formou, bývá zvykem posílat originály dopisů jejich konkrétním adresátům, a kopie těchto dopisů tzv. na vědomí jiným osobám (nejčastěji osobám nadřízeným). Také v elektronické poště existuje obdobná možnost: každá zpráva může mít jednoho či několik adresátů, ale kromě toho může být poslána "na vědomí" jednomu či několika dalším uživatelům, ve formě **kopie** (anglicky: **Carbon Copy**, zkratkou CC).

Kopie zprávy se od originálu nijak neliší. Je v ní vždy uveden "hlavní" příjemce (resp. příjemci), zatímco příjemci kopií jsou v hlavičce uvedeni také, ale v rámci samostatné položky, například:

From: Jiri Peterka <peterka@kki.ms.mff.cuni.cz>

To: Jan Pavelka <pav@dcit.cz>

Cc: správce sítě <admin@dcit.cz>

Příjemce "kopie na vědomí" tedy má možnost dozvědět se, že příslušná zpráva je pouze kopií, a stejně tak má možnost zjistit, komu byl odeslán její originál, případně kterým dalším uživatelům byly odeslány další kopie - potřebné informace jsou v hlavičce obsaženy, a záleží na použitém programu pro zpracování pošty (realizujícím uživatelskou složku), zda a v jaké formě tyto informace svému uživateli poskytne.

Stejnou možnost má samozřejmě i příjemce originálu - také on se může dozvědět, že příslušná zpráva byla odeslána ve formě kopie jiným uživatelům.

To ovšem nemusí být v některých situacích příliš žádoucí. Proto existuje ještě jedna varianta "kopie na vědomí", a to tzv. **slepá kopie** (**Blind Copy**, zkratkou Bcc:). Liší se v tom, že o existenci této kopie se příjemce originálu (a většinou ani příjemce "běžné" kopie) nemá šanci dozvědět - v hlavičce zprávy, kterou obdrží, nejsou příjemci slepých kopií uvedeni.

Komfort poštovních programů

Možnost zasílání "kopií na vědomí" je příkladem mechanismu, který musí být podporován již samotným systémem přenosu zpráv (MTS, Message Transfer System), neboli soustavou vzájemně propojených přenosových složek (MTA, Message Transfer Agents), resp. protokolem, na základě kterého tyto složky pracují.

Existuje však také celá řada dalších "doprovodných" služeb, realizovaných výhradně těmi programy, které slouží uživatelům k bezprostřednímu zpracování elektronické pošty (tedy uživatelskými složkami, User Agents).

Obvyklou samozřejmostí bývá vedení všelijakých adresářů, ve kterých si koncoví uživatelé zaznamenávají elektronické adresy svých přátel, kolegů, spolupracovníků, partnerů apod. Další užitečnou funkcí je možnost vytváření distribučních seznamů, citovaná výše (pro potřeby rozesílání zpráv více příjemcům).

Tomu, kdo elektronickou poštu využívá intenzivněji a má bohatou korespondenci, přijde velmi vhod možnost zařazovat jednotlivé zprávy do oddělených "fochů" (pomyslných přihrádek, škatulek apod., anglicky: **folders**). Takto si například může shromažďovat na jednom místě (v jednom "folderu") korespondenci s jedním konkrétním uživatelem, zatímco do jiného "folderu" si bude ukládat poštu, týkající se určitého konkrétního tématu apod.

Zařazování pošty do jednotlivých "folderů", které si uživatel vytváří podle své potřeby, bývá nejčastěji řešeno ručně. Některé poštovní programy však nabízí i možnost automatického třídění, například na základě adresy odesilatele, na základě výskytu určitého klíčového slova v předmětu zprávy apod.

Velmi praktická bývá i možnost přímé odpovědi, spojená s vkládáním citací z původního dopisu. Pokud se například při čtení určité zprávy rozhodnete na ni okamžitě odpovědět (což bývá dobrým zvykem), pak většinou stačí zmáčknout jedinou horkou klávesu, a váš poštovní program vám připraví základ odpovědi: sám si například vyplní adresu příjemce (kterou si odvodí z přijaté zprávy), sám si zvolí předmět zprávy (většinou vezme původní předmět, a před něj připíše výmluvné Re:). Kromě toho však může vložit do textu odpovědi také původní text (uvozený nějakým speciálním znakem, obvykle pravou špičatou závorkou), a vám pak stačí jen vhodně připisovat své odpovědi. Například:

> Kdy uz konecne napises dalsi dil serialu "Co je cim ..

> v pocitacovych sitich?"

Uz se na tom pracuje.

> A o cem to bude?

No porad jeste o elektronicke poste

---

opět prosím o šetrné zalámání, ať není porušen smysl předchozí ukázky

---

Přílohy ke zprávám

Elektronická pošta je určena především k přenosu zpráv, které mají textovou podobu, a o kterých se předpokládá, že nejsou příliš velké. S textovým charakterem zpráv pak

souvisí i to, že přenosové složky, které zprávy skutečně přenášejí, se na ně dívají jako na posloupnost znaků, a jako takové je i přenáší.

To má ovšem některé významné důsledky: například ten, že příslušný přenosový protokol může vycházet z předpokladu, že půjde výhradně o anglický text, a že tedy jednotlivé znaky může znázornit jen v sedmi bytech (čehož pak využije pro efektivnější přenos). Pokud tak skutečně činí, pak není možné používat v textu zpráv například naše proslavené háčky a čárky.

Dalším důsledkem, který vyplývá z textového charakteru zpráv, je nemožnost přenášet tímto způsobem také binární soubory, které je nutné chápat nikoli jako posloupnost znaků, ale jako posloupnost jednotlivých bitů. Za binární soubory je nutné považovat například také soubory ve vnitřním formátu různých textových editorů a procesorů.

V praxi by se ale takováto možnost velmi hodila - kdybychom například mohli připojit k odesílané zprávě jako přílohu binární soubor. Některé systémy elektronické pošty této možnosti vychází vstříc: umožňují připojovat ke zprávám přílohy (attachments), které mohou být i binárními soubory. S textovým charakterem přenosu se pak vyrovnávají tím, že tyto binární soubory pro potřeby přenosu vhodně zakódují (tj. převedou do "čistě textového" tvaru), a při příjmu je zase převádí zpět do jejich původního tvaru.

Takže až budete někomu příště psát elektoronickou poštou, můžete mu ke své zprávě "přibalit" například hezký obrázek ve formě binárního souboru. Zase ale jenom tehdy, pokud váš poštovní program (User Agent) tuto možnost podporuje, a stejně tak i poštovní program na straně adresáta.

83) Elektronická pošta v prostředí TCP/IP sítí

V minulých třech dílech seriálu jsme se věnovali uživatelskému pohledu na elektronickou poštu - na její celkovou filosofii a funkční možnosti. Přitom jsme k celé problematice přistupovali obecně, a snažili se ukázat to, co je pro všechny systémy elektronické pošty společné. Nyní se již zaměříme na jednu konkrétní oblast a na způsob, jakým je elektronická pošta řešena právě v ní: na prostředí počítačových sítí na bázi TCP/IP.

Nejprve si ale znovu připomeňme základní představu o způsobu implementace elektronické pošty, kterou jsme si zavedli již v 81. dílu: že jednotlivé zprávy si mezi sebou vyměňují tzv. *přenosové složky* (MTA, Message Transfer Agent), které jsou pro běžného uživatele neviditelné. Ten naopak komunikuje s tzv. *uživatelskými složkami* (UA, User Agent), jejichž prostřednictvím čte došlé zprávy, sestavuje nové a zadává je k odeslání. Jak jsme si již také uvedli v 81. dílu, tyto dvě složky mohou, ale nemusí být provozovány na témže počítači.

Odlišnosti v konvencích a přenosových mechanismech

V minulých dílech jsme si také řekli, že existují různé systémy elektronické pošty, které v obecném případě používají různé konvence a různé přenosové mechanismy pro přenos jednotlivých zpráv. Tyto konvence a mechanismy přitom nemusí být slučitelné s obdobnými konvencemi a mechanismy, které používají jiné systémy elektronické pošty. Přesto je ale možné, aby si i různé systémy dokázaly vzájemně předávat poštu - jsou-li vhodně propojeny pomocí tzv. *poštovních bran* (mail

gateways), které zajišťují potřebné konverze. Přitom je ale možné i to, aby v rámci jednoho a téhož systému elektronické pošty spolu úspěšně a přímo komunikovaly (v roli přenosových složek) i velmi různorodé počítače, lišící se například systémovou platformou a operačním systémem apod. Co je tedy vlastně určujícím pro možnost přímé komunikace mezi dvěma přenosovými složkami MTA, a kdy takováto komunikace dvou složek vyžaduje zprostředkovací funkce poštovní brány?

Odpověď je následující: nezáleží na způsobu implementace složek MTA (a tím ani na prostředí, ve kterém jsou provozovány). Záleží naopak na následujících skutečnostech:

- na způsobu, jakým složky MTA vzájemně komunikují, resp. jakým si předávají jednotlivé zprávy. Tento způsob je definován příslušným protokolem, označovaným obecně jako *protokol přenosu zpráv* (mail transfer protocol).

- na způsobu, jakým složky MTA interpretují obsah jednotlivých zpráv - zejména jejich hlaviček (viz minule), které mj. definují odesilatele i koncového příjemce zprávy (zatímco interpretace vlastního obsahu zpráv je vesměs ponechávána na příjemci). Závaznou interpretaci zpráv (jejich hlaviček) pak určuje standard, který přesně definuje formát jednotlivých zpráv, především pak syntaxi a sémantiku jednotlivých položek hlaviček.

Existence poštovní brány je nevyhnutná v případě, kdy komunikující složky používají odlišný formát zpráv. V případě, kdy používají stejný formát přenášených zpráv, ale liší se v používaném protokolu pro jejich přenos, potřebují mezi sebou také vhodného prostředníka. Je pak ovšem spíše terminologickou otázkou, zda takovýto prostředník bude označován jako poštovní brána či nikoli.

Tím, co je charakteristické pro systémy elektronické pošty, používané v sítích na bázi protokolů TCP/IP, je používání jednotného protokolu pro přenos zpráv (protokolu SMTP), a jednotného formátu zpráv (definovaného doporučením RFC 822).

SMTP - protokol přenosu zpráv

Protokol SMTP (od: Simple Mail Transfer Protocol) je novější verzí staršího protokolu MTP (Mail Transfer Protocol), který se ukázal jako zbytečně komplikovaný, a byl výrazně zjednodušen (odsud přívlastek "Simple" v názvu SMTP). Definuje pouze způsob, jakým si jednotlivé složky MTA zprávy vyměňují, ale nikoli již to, jakým způsobem s nimi dále nakládají - například jakým způsobem jsou přijaté zprávy předávány uživatelům (příjemcům) apod. Dále protokol SMTP nedefinuje způsob a místo uchovávání jednotlivých zpráv (v poštovních schránkách uživatelů, případně ve frontách zpráv k odeslání apod.), ani to, kdy a jak často se má pokoušet zprávy odesílat. Obvykle je tento protokol implementován jako proces, který je explicitně volán jinými procesy, implementujícími složku MTA.

Podrobnějším popisem tohoto protokolu se budeme zabývat v samostatném dílu tohoto seriálu.

RFC 822 - standard pro formát jednotlivých zpráv

Jednotný formát zpráv elektronické pošty je definován v doporučení RFC (Request For Comment) číslo 822, s názvem: Standard for the Format of ARPA Internet Text Messages.

Tento standard rozlišuje dvě základní části zprávy - hlavičku (header) a tělo (body), a přesně definuje syntaxi i sémantiku hlavičky, resp. jejích jednotlivých částí, obsahující údaje relevantní pro doručování zpráv: mj. odesilatele a příjemce zprávy, její předmět (subject), datum a čas odeslání apod. V to pak spadá mj. i způsob konstrukce a zápisu adres, používaných pro potřeby elektronické pošty.

Pokud jde o tělo zprávy, zde standard předpokládá pouze to, že jde o text z ASCII znaků.

Standard, daný doporučením RFC 822, je značně rozšířený, a používá se i v sítích, které nepoužívají protokoly TCP/IP (zejména pak protokol SMTP pro vlastní přenos zpráv). Také většina alternativních standardů, které nejsou zcela slučitelné s RFC 822, mu jsou alespoň dosti podobné. Díky tomu je pak výrazně usnadněna výměna zpráv mezi těmito systémy, resp. konstrukce potřebných přestupních článků (poštovních bran).

Také standardem RFC 822 se budeme ještě podrobněji zabývat.

MIME - snaha odstranit omezení

Protokol SMTP i standard RFC 822 vznikly přibližně před dvanácti lety, jako výsledek určitého historického vývoje (úspěchu či neúspěchu předchozích protokolů), a také jako důsledek tehdejších potřeb - ty byly, pokud šlo o elektronickou poštu, zaměřeny hlavně na přenos jednoduchých ASCII textů. Tomu pak také byly oba standardy uzpůsobeny: RFC 822 hovoří jen o tom, že tělo zprávy je tvořeno ASCII textem, a protokol SMTP říká, že jednotlivé ASCII znaky mají být sedmibitové, a pokud jsou přenášeny takovou přenosovou cestou, které přenáší jednotlivé znaky jako osmibitové, pak zmíněných sedm bitů má být zarovnáno doprava, a nejvyšší bit má být nastaven na nulu.

Díky tomu se ovšem prostřednictvím pošty na bázi SMTP a RFC 822 nedá přenášet nic jiného, než jednoduché ASCII texty: například žádné binární soubory, žádné formátované texty (které by obsahovaly speciální formátovací znaky) či texty v národních abecedách, které vyžadují kódování jednotlivých znaků v osmi bitech. Jisté řešení se sice našlo - zakódovat binární data (např. pomocí tzv. UUENCODE-ování) do znakové podoby (tj. tak, aby výsledkem byly jen samé tisknutelné ASCII znaky, zobrazitelné v sedmi bitech). To je ale skutečně jen nouzové řešení, navíc spojené s mnoha dalšími problémy.

Časem proto vznikl velký tlak na dodefinování standardů pro elektronickou poštu v rámci protokolů TCP/IP tak, aby zvládaly i přenos jiných druhů dat, než jen ASCII textů. Tímto rozšířením se stal standard **MIME** (MultiPurpose Internet Mail Extensions), definovaný doporučením RFC 1341 (z roku 1992).

Tento standard je v jistém smyslu ortogonální k doporučení RFC 822, protože sám nemění strukturu hlavičky (kterou určuje RFC 822), ale definuje formát těla zprávy tak, aby toto mohlo být dále děleno na části (doposud nebylo tělo zprávy nijak dále strukturováno), a aby každá takováto část mohla nezávisle na ostatních obsahovat i jiné druhy dat, než jen jednoduché ASCII texty: například formátované texty, texty v nejrůznějších národních abecedách, digitální zvukové záznamy, grafické soubory a všechny ostatní druhy binárních souborů. Také tímto standardem se budeme podrobněji zabývat.

POP - možnost příjmu pošty na dálku

Jednou z charakteristických vlastností protokolu SMTP je doručování jednotlivých zpráv přímo koncovým příjemcům (přesněji počítačům, na kterých se nachází poštovní schránky příjemců), a nikoli jejich postupný přenos přes různé mezilehlé uzly: před přenosem každé jednotlivé zprávy odesílající strana vždy nejprve naváže přímé spojení s přijímající stranou, a teprve pak zprávu přenese.

Toto řešení ovšem vychází z předpokladu, že přijímající počítač je trvale v provozu (i když na straně odesilatele jsou pro případ potřeby implementovány fronty dosud neodeslaných zpráv, ve kterých tyto mohou určitou dobu čekat). Z tohoto důvodu jsou pak v roli poštovních uzlů využívány výhradně takové počítače, které jsou provozovány trvale - typicky různé Unixovské servery, a nikoli pracovní stanice, na kterých uživatelé skutečně pracují (a které nejsou trvale v provozu). Pro většinu uživatelů však není příliš výhodné zpracovávat si svou poštu přímo na takovémto počítači - zejména kvůli komfortu, který nabízí příslušné programy, realizující tzv. uživatelské složky (UA, User Agent). Jak jsme si naznačili již v 81. dílu, je v zásadě možné provozovat uživatelskou složku i na jiném počítači, než který je skutečným příjemcem pošty a na kterém se nachází poštovní schránky jednotlivých uživatelů. K tomuto účelu ovšem bylo nutné vyvinout také potřebný přenosový protokol, který by zprostředkovával komunikaci mezi přenosovou složkou (v roli poštovního serveru) a uživatelskou složkou (v roli klienta). Takovýchto protokolů vzniklo dokonce víc, přičemž tím nejpoužívanějším je dnes zřejmě protokol POP3 (Post Office Protocol, verze 3), sloužící pro přenos zpráv směrem k uživatelské složce (zatímco v opačném směru je obvykle využíván protokol SMTP, viz též obrázek 83.1). Také protokolem POP3 se v tomto seriálu budeme zabývat podrobněji. .pa

.PI OBR83_1.TIF, 20, 40, 10

Obr. 83.1.: Představa protokolů elektronické pošty

84/ RFC822

V minulých dílech tohoto seriálu jsme se seznámili s obecnými principy elektronické pošty a začali se zabývat tím, jak konkrétně bývá realizována v sítích na bázi protokolů TCP/IP. V minulém dílu jsme dospěli k představě toho, jaký je současný stav v této oblasti a jaké protokoly, resp. standardy a doporučení se používají v jednotlivých částech systémů elektronické pošty. Dnes se již dostaneme podrobněji k doporučení RFC822, které definuje formát přenášených zpráv.

Nejprve si ale připomeňme samotný statut dokumentu, označovaného jako RFC822 - jak jsme si již uvedli v předchozích dílech, dokumenty RFC (Request For Comment) jsou základními technickými dokumenty, které se zabývají nejrůznějšími aspekty fungování Internetu, včetně protokolů TCP/IP. Nemají formu právně závazných norem (proto je asi na místě označovat je spíše jako doporučení). Některé z nich - zvláště ty, které definují různé protokoly - však mají povahu skutečných a závazných standardů, protože jak v rámci Internetu, tak i mimo něj (a to i v komerční sféře) jsou všeobecně uznávány, respektovány a důsledně dodržovány. Dokument RFC822 je právě jedním z takovýchto dokumentů, které mají "sílu" závazného standardu.

Čeho se RFC822 týká?

Již v minulém dílu jsme si naznačili, že při přenosu jednotlivých zpráv elektronické pošty mezi přenosovými složkami je třeba uvažovat jak konkrétní mechanismus, kterým si tyto složky zprávy předávají, tak i samotný vnitřní formát těchto zpráv (kterému zmíněné přenosové složky musí rozumět alespoň do té míry, aby dokázaly poznat, komu a kam mají zprávy poslat).

Nejlépe je vyjít z představy, že každá jednotlivá zpráva má povahu listu papíru, a je přenášena v obálce. Na samotném listu papíru je jak vlastní obsah zprávy (tzv. tělo), tak i její hlavička, určující mj. odesilatele, konečného adresáta a obsahující i další údaje, se kterými se ještě dnes seznámíme. Tento list papíru přitom vyplňuje tzv. uživatelská složka, resp. program, jehož prostřednictvím uživatel sestavuje své zprávy. Když tato uživatelské složka (User Agent) předá připravenou zprávu (list papíru) přenosové složce ke skutečnému odeslání, ta vloží list do obálky, a na ni nadepíše takové údaje, jaké jsou zapotřebí pro přenos obálky (načež zajistí její skutečný přenos). Při sestavování těchto údajů přitom přenosová složka vychází z toho, co je napsáno na listu papíru, přesněji v hlavičce zprávy (konkrétně z údajů o odesilateli a příjemci atd.). Sama proto musí rozumět formátu, podle kterého je zpráva sestavena, resp. vyplněn náš hypotetický list papíru. No a právě tento formát je předmětem doporučení RFC822, zatímco nápisy na pomyslné obálce a její vlastní přenos se řídí protokolem SMTP (který je sám definován v doporučení RFC821). Doporučení RFC822 také nedefinuje způsob komunikace mezi tím, kdo zprávu sestavuje (uživatelskou složkou) a tím, kdo ji skutečně odesílá (přenosovou složkou). Dokonce ani nepředepisuje, jakým způsobem má přenosová složka jednotlivé zprávy uchovávat apod. Jediné, co definuje, je formát samotných zpráv (našeho pomyslného listu papíru), a to navíc ještě jen pro potřeby přenosu mezi jednotlivými přenosovými složkami.

Hlavička a tělo zprávy

Doporučení RFC822 chápe zprávu jako text, členěný na jednotlivé řádky, a rozděluje ji do dvou základních částí: na **hlavičku** (**header**) a **tělo** (**body**). Dále se podrobněji zabývá syntaxí a semantikou hlavičky, zatímco obsah těla zprávy nijak přesněji nevymezuje - pouze říká, že hlavička musí předcházet tělu, a tělo musí být od hlavičky odděleno nejméně jednou prázdnou řádkou (neboli: vše, co následuje za první prázdnou řádkou, je považováno za tělo zprávy).

Na hlavičku zprávy se doporučení RFC822 dívá jako na posloupnost položek, kterým v originále říká **header fields** (doslova: pole hlavičky). Každá položka musí začínat na nové řádce (a dokonce na první pozici řádky), a může pokračovat i na dalších řádkách (v takovém případě ale nikoli od první pozice).

Každá položka je vždy uvozena určitým klíčovým slovem (zakončeným dvojtečkou), které definuje její význam (a tím současně ulehčuje práci programům, které mají hlavičku analyzovat a řídit se jejím obsahem). Za tímto uvozujícím klíčovým slovem (a povinnou dvojtečkou) pak následuje vlastní obsah příslušné položky.

Některým položkám hlavičky doporučení RFC822 předepisuje určitou povinnou syntaxi, zatímco jiným nikoli. Logika je zde taková, že obsah některých položek analyzují programy, které se na přenosu zpráv podílí, a povinná syntaxe jim v tom velmi účinně pomáhá. Týká se to především položek, které obsahují adresy a údaje o datumu a čase. Naproti tomu u jiných položek (například u položky Subject, vyjadřující předmět zprávy) prakticky nepřipadá v úvahu, že by je bylo potřeba

analyzovat, a tak jejich syntaxe není nijak předepsána - mohou to tedy být libovolné texty.

Doporučení RFC822 také nepředepisuje povinné pořadí jednotlivých položek hlavičky (pouze vyslovuje určitá doporučení). Příklad hlavičky, sestavené podle RFC822, ukazuje obrázek 84.1.

Položky hlavičky

Samotné doporučení RFC822 definuje poměrně velký počet různých položek pro hlavičky zpráv. Cílem tohoto příspěvku samozřejmě není podat jejich vyčerpávající seznam či dokonce popis - od toho je samotný dokument RFC822, který je volně dostupný a šiřitelný (stejně jako všechny ostatní dokumenty RFC). My si zde pouze naznačíme některé nejčastější položky a ukážeme si, jak jejich skladba koresponduje s možnostmi, které nabízí dnešní systémy elektronické pošty.

Každá zpráva samozřejmě musí od někoho pocházet - někdo musel mít zájem na jejím odeslání a sestavil také její vlastní obsah (tělo zprávy). Doporučení RFC822 na tuto osobu pamatuje položkou "From", například:

From: pet@dcit.cz

tato položka je přitom jednou z těch, u kterých je předepsána povinná syntaxe. Ta však stále nabízí řadu variant, včetně možností vkládání komentářů, srozumitelných člověku. Například:

From: pet@dcit.cz (Jiri Peterka)

From: Jiri Peterka <pet@dcit.cz>

From: "Peterka, Jiri" <pet@dcit.cz>

apod. Doporučení RFC822 dále pamatuje i na možnost, že původní iniciátor zprávy (nebo lépe: autor zprávy) není totožný s jejím skutečným odesilatelem. K takovéto situaci může dojít například tehdy, když někdo posílá vzkaz někoho jiného, když si příliš zaneprázdněný šéf nechává odesílat zprávy svou sekterářkou, nebo také například tehdy, když někdo pošle příspěvek do některé z elektronických konferencí. Pak jej totiž pošle nejprve na adresu správce konference, a ten ji následně rozešle všem účastníkům konference. Ti pak dostanou zprávu, jejímž odesilatelem je správce konference (ať již je to člověk či pouhý program), zatímco autorem zprávy je někdo úplně jiný. RFC822 se s tím vyrovnává zavedením položky "Sender", která specifikuje odesilatele zprávy (je-li to někdo jiný, než autor zprávy, pro kterého je určena položka "From").

Další položkou, která souvisí s možnou disproporcí mezi autorem a odesilatelem zprávy, je položka "Reply-To". Ta totiž explicitně říká, kam mají být zasílány odpovědi,. Například právě u elektronických konferencí je dosti podstatný rozdíl mezi tím, když se odpověď pošle na adresu konference (a pak ji dostanou všichni její účastníci), nebo když se pošle přímo původnímu autorovi (a pak ji dostane pouze on). Pomocí položky "Reply-To" lze explicitně předepsat, kam mají být odpovědi posílány. Je to užitečné například i pro uživatele, kteří mají více pracovišť, resp. svou elektronickou poštu odesílají z více různých míst - pomocí této položky si pak mohou předepsat, že odpovědi jim mají chodit na jedno konkrétní místo. Ještě další možnost nabízí položka "Return-Path", která umožňuje předepsat, kam má být zpráva vrácena v případě její nedoručitelnosti.

Podobnou syntaxi, jako položka "From" (i další, které jsme zmňovali v předchozím odstavci), má i položka "To", určující adresáta zprávy. Jak jsme si již uvedli v předcházejících dílech, zprávy elektronické pošty je možné zasílat i jako tzv. kopie na vědomí (Carbon Copy), a dokonce i jako tzv. "slepé" kopie (Blind Carbon Copy) - o kterých se hlavní adresát zprávy nedozví (na rozdíl od příjemců běžných kopií, o kterých se v hlavičce svého originálu dozví). Pro specifikaci adresátů kopií jsou určeny položky "Cc" a "Bcc", například:

From: Jiri Peterka <pet@dcit.cz> (odesilatel

To: Jan Novak <novak@cuni.cz> (adresát)

Cc: Petr Votava <votava@dcit.cz> (příjemce kopie)

Bcc: Jan Pavelka <pav@dcit.cz> (příjemce slepé kopie)

V každé přijaté zprávě bývá obvykle obsažen určitý počet položek "Received", které obsahují záznamy o "cestě" zprávy mezi jednotlivými přenosovými složkami - v zásadě by šlo říci, že každému jednotlivému "přeskoku" by měla odpovídat jedna tato položka (a jedna další obvykle připadá i na první přenosoovu složku, která zprávu příjme k odeslání). V položkách "Received" pak může být vyjádřeno například i to, jakým protokolem byla zpráva přenesena, pod jakým označením apod. - tyto informace jsou většinou určeny pro potřeby ladění a řešení nestandardních situací. Například:

Received: from einar.dcit.cz by rifflet.ibp.fr with SMTP (1.38.193.4/16.2) id AA02305; Wed, 20 Jul 1994 13:56:26 +0200 Received: from kaare.dcit.cz by einar.dcit.cz (AIX 3.2/UCB 5.64/4.03) id AA08541; Wed, 20 Jul 1994 13:56:12 +0200 Received: from frode.dcit.cz by kaare.dcit.cz (5.0/SMI-SVR4) id AA01366; Wed, 20 Jul 1994 13:56:56 +0200

Další položky, na které RFC822 pamatuje, umožňují vyjádřit datum a čas, kdy byla zpráva zadána k odeslání (položka "Date"), její předmět (položka "Subject") apod. Dále RFC822 pamatuje například i na možnost automatického přesměrovávání (tzv. auto-forward), kdy uživatel sice přijímá poštu na určité adrese, ale odsud si ji nechává automaticky posílat ještě někam jinak - což je výhodné například tehdy, když na určitou dobu odcestuje někam, kde je stále v dosahu elektronické pošty, a chce nadále přijímat svou běžnou elektornickou korespondenci. Doporučení RFC822 zavádí za tímto účelem celou sérii položek, začínajících prefixem "Resent-", například:

Resent-From: pet@dcit.cz (odkud byla přesměrovaná zpráva odeslána)

Resent-To: peterka@rifflet.ibp.fr (kam ....)

Resent-Date: Wed, 20 Jul 94 13:52:34 MET-1DST (kdy .....)

Adresy á la RFC822

Další velmi důležitou součástí doporučení RFC822 je i přesná specifikace možných zápisů adres příjemců i odesilatelů, kterou RFC822 definuje v rámci předepsané syntaxe těch položek, které tyto adresy obsahují (např. položky "From", "To","Sender" apod.). Možné způsoby zápisu adres přitom vychází z existence hierarchicky uspořádaných domén a tomu odpovídajících pravidel pro tvorbu adres, které jsou používány nejen v rámci Internetu (a kterými jsme se zabývali již v 80. dílu

tohoto seriálu). Jelikož jejich přesná syntaxe je specifikována právě doporučením RFC822, jsou tyto adresy označovány často také jako "RFC822 adresy".

Možnosti dalšího rozšiřování

Repertoár položek (header fields), které doporučení RFC822 zavádí, není chápán jako pevně daný a nerozšiřitelný. Počítá naopak s tím, že další položky budou zaváděny postupně (jako tzv. "extension-fields"), a budou standardizovány stejným způsobem, jako samotné doporučení RFC822 (tj. formou dokumentů RFC).

Kromě toho ale RFC822 počítá i s tím, že také uživatelé si mohou chtít zavádět své více či méně "lokální" položky, sledující potřeby jejich konkrétních poštovních systémů. I na takovéto uživatelsky definované položky je pamatováno, a je pouze požadováno, aby jejich identifikátory (uvozující položku) nekolidovaly s jinými (standardizovanými) položkami. Za tímto účelem je v doporučení RFC822 stanoveno, že žádné "oficiální" rozšiřující položky, které budou kdy zavedeny, nebudou začínat řetězcem "X-". Ten je naopak doporučen jako začátek všech uživatelských rozšíření. Například položka:

X-Mailer: ELM (version 2.2 PL16 mips 1)

napovídá o tom, jaký poštovní program byl použit pro sestavení zprávy. Systémy, které takovouto položku (obecně jakoukoli uživatelskou položku) dokáží rozpoznat a správně interpretovat, ji mohou využít - ostatní ji chápou jednoduše jako komentář a ignorují ji.

Obr. 84.1: Příklad hlavičky zprávy dle RFC822

From pav@dcit.cz Wed Jul 20 16:23 MET 1994 (jméno odesilatele s údajem o době doručení)

Received: from einar.dcit.cz by rifflet.ibp.fr with SMTP (počítač einar.dcit.cz předal tuto zprávu) *(1.38.193.4/16.2) id AA02512; Wed, 20 Jul 1994 16:23:15 +0200* (počítači rifflet.ibp.fr) *Return-Path: <pav@dcit.cz>* (údaj o zpáteční cestě k autorovi dopisu) *Received: by einar.dcit.cz (AIX 3.2/UCB 5.64/4.03)* (počítač einar. dcit.cz převzal tuto zprávu) *id AA22996; Wed, 20 Jul 1994 16:22:23 +0200* (k odeslání)

Date: Wed, 20 Jul 1994 16:22:23 +0200 (datum a čas, kdy byla zpráva předána k odeslání)

From: "Jan Pavelka" <pav@dcit.cz> (odesilatel) *Message-Id: <9407201422.AA22996@einar.dcit.cz>* (identifikátor zprávy)

Reply-To: pav@dcit.cz (kam mají být odesílány odpovědi) *To: peterka@rifflet.ibp.fr* (adresát) *Subject: Jak je ve Francii?* (předmět zprávy)

85/ SMTP

V minulém dílu tohoto seriálu jsme se zabývali základním formátem zpráv elektronické pošty, tak jak jej v rámci síťového modelu TCP/IP definuje doporučení RFC 822. Dnes se již dostaneme k představě toho, jak jsou takto sestavené zprávy přenášeny.

Nejprve se ale vraťme k představě, kterou jsme si zavedli již v minulém dílu, a která nám bude velmi užitečná i dnes. Jde o představu zprávy, napsané na listu papíru a vložené do obálky - to, jak konkrétně má být zpráva na listu papíru napsána, definuje doporučení RFC822 (kterým jsme se zabývali minule). To, jak má vypadat obálka, co má být na ní napsáno a jakým způsobem se má přenášet, je definováno protokolem SMTP, kterým se budeme zabývat dnes.

Na světě není jen SMTP

Hned na úvod je dobré si zdůraznit, že protokol SMTP (Simple Mail Transfer Protocol) je ve své podstatě přenosový prostředek, resp. mechanismus, koncipovaný s ohledem na potřeby přenosu zpráv. Není ovšem zdaleka jediným takovýmto přenosovým prostředkem, resp. mechanismem. Je pouze tím, který je nejrozšířenější v jednom druhu sítí (sítí na bázi TCP/IP), ale ani zde nemá zcela výlučné postavení. Jeho předchůdcem, který vznikl ještě mimo rámec síťového modelu TCP/IP, ale dodnes se i v něm stále ještě používá pro přenos elektronické pošty, je protokol UUCP (Unix to Unix Copy Protocol).

Možnosti, které protokol SMTP nabízí, přitom nejsou omezeny jen na přenos a doručování zpráv do poštovních schránek uživatelů, tak jak jsme až dosud při našich úvahách o elektronické poště předpokládali. Jelikož vznikl již před určitým časem - poprvé byl publikován ve formě doporučení RFC788 v listopadu roku 1981, a v srpnu 1982 byl "novelizován" doporučením RFC822 - pamatuje protokol SMTP i na existenci uživatelů, pracujících na terminálech hostitelských počítačů: těm, kteří jsou momentálně přihlášeni na některém z terminálů hostitelského počítače, je připraven příslušnou zprávu také okamžitě zobrazit (buď místo jejího uložení do uživatelovi schránky, nebo obojí). Tato možnost, na kterou protokol SMTP pamatuje, má však dnes již spíše historický význam, a v praxi se nepoužívá.

SMTP obálka

Na pomyslné obálce, do které je pro potřeby přenosu vkládána vlastní zpráva (sestavená dle doporučení RFC822), musí vždy "nadepsány" alespoň dva základní údaje - adresa uživatele, na jehož popud byla obálka se zprávou sestavena (tzv. *originator*), a údaj o tom, komu je zasílána (tzv. *recipient*). Je přitom možné i to, aby jedna a tatáž zpráva měla více příjemců, resp. byla přenášena jen jednou, ale v místě svého příjmu byla doručena více příjemcům. Další informace, jako například předmět zprávy či datum a čas jejího odeslání, jsou již pouze součástí zprávy samotné.

Na tomto místě je velmi důležité si uvědomit, že adresa příjemce (recipient-a) na SMTP obálce vůbec nemusí být shodná s adresátem vlastní zprávy. Důvodů k tomu je hned několik - například ten, že jedna a tatáž zpráva může mít kromě svého "hlavního" příjemce (vyjádřeného v hlavičce zprávy v položce To:) také jednoho či několik příjemců běžných či slepých kopií (vyjádřených v položkách Cc:, resp. Bcc:). Když je pak zpráva doručována těmto příjemcům kopií, je na obálce "nadepsána" jejich adresa, zatímco v položce To: v rámci vlastní zprávy zůstává nadále adresa hlavního adresáta. Analogicky je tomu i v případě, kdy "hlavních" adresátů (uvedených v položce To: hlavičky zprávy) je více, v případě zasílání jedné zprávy celému seznamu uživatelů apod. Vedle toho však existuje ještě celá řada dalších důvodů, kvůli kterým se adresy na SMTP obálce mohou lišit od adres, uvedených v hlavičce vlastní zprávy. Tyto důvody souvisí s konkrétním využitím protokolu SMTP v prostředí sítě Internet a s používáním systému doménových jmen v této soustavě

sítí. Ve svém konečném efektu tyto důvody vedou k tomu, že jednotlivé zprávy mohou být na své cestě ke konečnému adresátovi (vyjádřeném v položce To: hlavičky zprávy) postupně přenášeny přes různé mezistupně (přes různé *recipent-y)*. Této problematice se budeme podrobněji věnovat v některém z dalších pokračování seriálu.

Přenos obálky a jejího obsahu

Protokol SMTP předpokládá, že "obálka" i její obsah budou přenášeny po takovém transportním spojení, které se samo postará o zajištění spolehlivosti přenosu. Obvykle je toto transportní spojení zajišťováno protokolem TCP (v sítích na bázi TCP/IP prakticky výlučně), ale v zásadě není vyloučeno ani použití jiných přenosových protokolů, zajišťujících spolehlivý přenos. Existuje například doporučení RFC1090, definující způsob provozování SMTP nad protokoly X.25.

Protokol SMTP chápe přenášená data jako textová, členěná na jednotlivé řádky (pomocí znaků CR a LF), a tvořená pouze znaky z původní 128-prvkové abecedy ASCII. Jinými slovy: SMTP předpokládá pouze přenos znaků, kódovaných do sedmi bitů. Pokud se takovéto sedmibitové znaky přenáší kanálem, který je uzpůsoben přenosu osmibitových znaků resp. bytů (což je mj. příklad protokolu TCP), pak standard SMTP definuje, že jeho sedmibitové znaky mají být vkládány do osmic bitů tak, aby byly zarovnány doprava a zleva doplněny nulovým bitem.

Pomyslná obálka protokolu SMTP se svými údaji je přitom přenášena také ve formě textu, a to na začátku přenosu. Celá komunikace mezi příjemcem a odesilatelem (po navázání spojení na úrovni protokolu TCP) má formu dialogu, v rámci kterého se obě strany nejprve informují o své obecné připravenosti přijímat poštu, pak si předají údaje o odesilateli a příjemci (resp. další údaje z pomyslné obálky), a poté pak i samotný samotný obsah zprávy.

Forma SMTP dialogu

Vlastní dialog obou komunikujících stran má formu předávání příkazů, a zasílání odpovědí na ně. Příkazy jsou tvořeny klíčovými slovy, za kterými obvykle následují další upřesňující parametry. Například příkazem

HELO einar.dcit.cz

sděluje odesílající uzel na začátku vzájemného dialogu svou identitu přijímajícímu uzlu, zatímco příkaz

MAIL FROM: <pet@dcit.cz>

signalizuje, kdo je odesilatelem zprávy, příkaz

RCPT TO: <pav@einar.dcit.cz>

zase specifikuje příjemce zprávy apod. Jak jsme si ale již uvedli výše, nemusí tyto adresy z SMTP obálky nutně odpovídat adresám prvotního odesilatele a konečného příjemce, uvedeným v hlavičce zprávy.

Naproti tomu odpovědi na příkazy jsou zásadně číselné, tvořené trojmístnými desítkovými čísly (přenášenými v textovém tvaru). Jejich struktura je přitom obdobná struktuře číselných odpovědí, používaných v protokolu FTP (kterými jsme se podrobněji zabývali v 72. dílu seriálu): první z trojice číslic (1 až 5) vyjadřuje

celkovou povahu odpovědi (např. 2 znamená kladnou odpověď, 5 zápornou odpověď), druhá číslice ji dále upřesňuje a konečně třetí ji specifikuje ještě blíže (např. odpověď 220 signalizuje celkovou připravenost druhé strany k přenosu pošty, odpověď 250 potvrzuje úspěšné splnění požadavku, zatímco například odpověď 550 říká, že požadovaná akce nemohla být provedena proto, že příslušný uživatel není na přijímající straně znám, resp. jeho poštovní schránka není k dizpozici, odpověď 552 signalizuje nemožnost dokončení požadované akce kvůli nedostatku paměti pro dočasné uložení apod.). Obdobně jako u protokolu TCP je i zde sledována potřeba různě "silných" analyzátorů - ty nejjednodušší dokáží vystačit jen s první číslicí, zatímco ty složitější se orientují i podle dalších číslic. Případné textové řetězce, kterými bývají číselné části odpovědi doplněny, pak mají pouze povahu komentářů (určených pro případné "lidské" příjemce), a jejich tvar není standardizován. Může se proto případ od případu lišit.

Fáze dialogu

První fází dialogu podle protokolu SMTP je samotné navázání spojení. Ten, kdo spojení navazuje (za účelem následného odeslání zprávy), vystupuje v roli klienta, a kontaktuje protistranu v roli SMTP serveru, který bude zprávu přijímat. Klient přitom kontaktuje svůj server na portu č. 25, který patří mezi tzv. dobře známé porty (well-known ports, viz 55. díl seriálu). Je-li vše v pořádku a server je připraven žádost přijmout, odpoví na ni číselnou odpovědí 220 (doplněnou případným textovým komentářem), viz řádek 1 obrázku 85.1. Klient na to reaguje tak, že se serveru představí - pomocí příkazu HELO, viz řádek 2 na obr. 85.1. Pokud je server ochoten přijímat poštu od tohoto klienta, odpoví na jeho představení kladně (odpovědí 250, viz 3. řádek).

Další fází vzájemného dialogu mezi klientem (odesílajícím) a serverem (přijímajícím) je přenos pomyslné obálky, spočívající v předání adresy odesilatele (příkazem MAIL FROM:, viz řádka č. 4) a jedné či několika adres příjemců (pomocí příkazu RCPT TO:, viz řádky 6 a 8).

Poté již následuje přenos vlastní zprávy. Ten je ze strany vysílajícího klienta uvozen příkazem DATA (viz řádek 10), a ze strany přijímajícího klienta odpovědí 354 (je-li server na příjem zprávy připraven, viz řádek 11). Samotná zpráva, tvořená svou hlavičkou a tělem (a struturovaná podle doporučení RFC822, viz minulý díl) je přenášena po jednotlivých řádcích. Konec zprávy je signalizován řádkou, obsahují pouze jediný znak - tečku (na první znakové pozici). Případný výskyt takovéhoto řádku v těle zprávy je řešen zdvojením uvozující tečky.

Po úspěšném přenosu celé zprávy následuje kladné potvrzení (odpovědí 250, viz řádka 13 obrázku 85.1), a ukončení spojení ze strany klienta (příkazem QUIT, viz 14. řádek).

.... následující text tvoří obrázek 85.1. .....

odesílající (klient) přijímající (server)

---------- ----------

{navázání spojení na úrovni protokolu TCP}

1) 220 einar.dcit.cz SMTP service ready

2) HELO frode.dcit.cz

3) 250 einar.dcit.cz hello frode.dcit.cz

4) MAIL FROM <pet@dcit.cz>

5) 250 sender ok

6) RCPT TO: <pav@einar.dcit.cz>

7) 250 recipient ok

8) RCPT TO: <votava@einar.dcit.cz>

9) 250 recipient ok

{přenos dat}

10) DATA

11) 354 Enter mail, end with "." on a line by itself

12) {hlavička zprávy dle RFC 822}

.....

{tělo zprávy dle RFC822}

.

13) 250 mail accepted {ukončení přenosu dat}

14) QUIT

15) 221 einar.dcit.cz closing connection

{ukončení spojení na úrovni protokolu TCP}

—J—

—K—

—L—

—M—

—N—

—O—

—P—

—R—

—Ř—

—S—

—Š—

—T—

—U—

—Ú—

—U—

—V—

—Z—

—Ž—

—Z—